Vers des mécanismes génériques de communication et une meilleure maîtrise des affinités dans les grappes de calculateurs hiérarchiques

Brice Goglin

Mémoire
présenté par

Brice Goglin

Chargé de recherche Inria Bordeaux - Sud-Ouest
Laboratoire Bordelais de Recherche en Informatique
Équipe projet commune Runtime

en vue de l'obtention du diplôme d'

Habilitation à Diriger des Recherches
de l'Université de Bordeaux
Spécialité : Informatique

Vers des mécanismes génériques de communication et une meilleure maîtrise des affinités dans les grappes de calculateurs hiérarchiques

Date de soutenance : 15 avril 2014

Composition du jury :

Président :	Jean Roman	Professeur, Institut Polytechnique de Bordeaux
Rapporteurs :	Christine Morin	Directeur de Recherche, Inria Rennes
	Franck Cappello	Senior Computer Scientist, Argonne National Lab.
	Vivien Quéma	Professeur, Institut National Polytechnique de Grenoble
Examinateurs :	Pascale Rossé-Laurent	Senior architect HPC, Bull SAS
	Jesper Larsson Träff	Professeur, Université de Technologie de Vienne

À mes parents.

Remerciements

Je te tiens tout d'abord à remercier chaleureusement les membres du jury, en particulier les rapporteurs du manuscrit, mais aussi Jesper qui s'est retrouvé seul étranger parmi tous ces français.

Un grand merci ensuite à Raymond qui a été régulièrement impliqué dans ma carrière. En effet, lors mon séjour temporaire du côté obscur (la physique théorique), son cours de Systèmes d'exploitation en 1999 puis ses conseils m'ont confirmé que je devais revenir à l'informatique. Puis il aura supporté le bruit du serveur qui me servait d'ordinateur de bureau (merci Loïc !) au début de mon stage de DEA. Ensuite il aura rapporté ma thèse, quoique ses balades en moto à Arcachon à l'été 2005 m'ont laissé me demander s'il rapportait effectivement assidûment :) Enfin il m'aura fait venir à Bordeaux et m'aura souvent conseillé depuis 8 ans. Merci également à Emmanuel qui nous a rejoint plus récemment mais m'a beaucoup aidé à préparer cette habilitation.

Merci également à tous les collègues. D'abord, mon co-bureau Guillaume, grand adepte du mangeage de fougasse qui empeste dans le bureau, et grand adepte du maigrir même si je n'ai déjà plus aucun gras pour me protéger du froid dans le bureau (merci la clim automatique du nouveau bâtiment Inria). Je suis sûr que toi aussi tu parviendras bientôt à éviter les bâtons qu'on nous met dans les roues... Ensuite Pierre-André qui m'a accueilli dans son bureau à mon arrivée à Bordeaux. Enfin tous les autres collègues de l'équipe. Un clin d'œil spécial à Nathalie qui sera toujours la bienvenue dans nos projets quand elle aura vraiment décidé de ne plus faire du portage sous Windows. Et un autre clin d'œil aux guignols de l'open-space et sur IRC, qui se reconnaîtront, qui m'ont bien détendu pendant la rédaction de ce manuscrit à l'été 2013.

Merci ensuite à mes anciens doctorants pour avoir fait de belles thèses. D'abord Stéphanie, partie dans les contrées lointaines du Tennessee. Puis Bertrand (sûrement parti à 100km d'ici à vélo pour la journée) avec qui j'ai eu la chance(?) de co-rédiger nos thèse et HDR respectives l'été dernier et qui a toujours été là quand il fallait me rappeler quels sports étaient des vrais sports. Un grand merci également aux stagiaires François (Broq), Romain, Antoine, Benoît et Nicolas. Et enfin à mon ex-ingé Ludovic S. pour toutes les discussions très intéressantes qu'on a partagées avant qu'il n'aille bosser pour la NSA française :) Ça a été cool de bosser avec vous tous.

Un coucou à tous les collègues plus ou moins étrangers, dans un ordre plus ou moins chronologique : Loïc qui m'aura transmis sa passion, Patrick, Marty et tous les autres de Myricom ; Dave et Darius d'Argonne ; Jeff de Cisco ; George, Thomas et Aurélien de UTK ; Josh de La Crosse (c'est quoi ce nom de ville étrange d'ailleurs ?) ; Marcelo de San Luis ; et d'autres que j'oublie. Et aussi les collègues moins étrangers, par exemple les Anne-Cécile, Eddy et Laurent, mes co-organisateurs du stand Inria à SuperComputing, qui vont, je l'espère, bientôt arriver à discuter avec la direction Inria sans que je ne serve de secrétaire :)

Un très grand merci à tous les collègues non-scientifiques du centre Inria Bordeaux - Sud-Ouest. Dans le désordre, les filles de la Com : je ne suis toujours pas certain de pourquoi c'est moi qui me suis retrouvé à bosser sur la médiation scientifique avec vous ; je ne suis toujours pas sûr non plus de tout comprendre à votre travail ; mais au moins j'apprends des trucs. Le SED (entretien et dépannage, même s'il serait bien qu'on sache un jour à quoi ça sert), toujours là pour filer un coup de main pour nos développements logiciels (si seulement leur direction nationale pouvait comprendre aussi bien nos problématiques ;-). Le SIC (moyens infos), du centre Inria mais aussi de PlaFRIM et du Mésocentre, qui supportent depuis 7 ans mes besoins bizarres et n'ont jamais rechigné à me filer un accès root pour mes recherches malgré mon

passif douteux. Et tous les autres que je croise plus ou moins régulièrement et avec qui j'aime discuter d'autre chose que de recherche et me changer les idées. Merci aussi à l'AGOS qui m'aura fait prendre l'air pour aller acheter le baby-foot pour préparer la prochaine coupe du monde.

Sur des aspects pratico-pratiques, merci à Bertrand et Sylvain pour les hacks en LaTeX dans le manuscrit, à croire que je vieillis trop pour les écrire moi-même. Merci à Emmanuel, Sylvain, Guillaume, Paul-Antoine, Bertrand, Cyril et Émilie pour les relectures de manuscrit, résumé et slides, à Lola et Ludovic C. pour les parties chelous qui les concernent dans la conclusion. Merci à Emmanuel, Guillaume, Nathalie, Ludovic et Lilia pour les répétitions de soutenance.

Enfin un très gros bisous à mon adorable femme Émilie qui a passé deux jours aux fourneaux pour préparer le pot, comme pour ma soutenance de thèse, et qui est toujours là pour moi. Heureusement qu'elle ne sous-estime pas autant que moi l'appétit des collègues. Et un gros bisous à mon fils Melvin qui a bien raison de me forcer à lâcher mon ordinateur pour aller jouer dans sa chambre ou dehors.

Une contrepèterie s'est cachée dans cette section, sauras-tu la retrouver ?

Résumés

Résumé

Avec l'utilisation de plus en plus répandue de la simulation numérique dans de nombreuses branches de l'industrie, le calcul haute performance devient essentiel à la société. Si des plates-formes de calcul parallèle de plus en plus puissantes continuent à être construites, leur utilisation devient cependant de plus en plus un casse-tête. En effet, leur complexité croît avec la multiplication des ressources de calcul et de stockage impliquées, leurs fonctionnalités hétérogènes, et leur répartition non-uniforme. De nouveaux outils logiciels sont nécessaires pour faciliter l'exploitation de ces plates-formes.

Je présente tout d'abord mes travaux visant à rendre plus accessibles et portables les mécanismes de communication développés par les constructeurs de réseaux haute performance pour le calcul. J'ai appliqué ce principe d'une part aux réseaux traditionnels de type Ethernet, et d'autre part aux communications entre processus locaux, afin d'améliorer les performances du passage de messages (MPI) sans dépendre de technologies matérielles spécialisées.

J'explique ensuite comment faciliter la gestion des calculateurs hiérarchiques modernes. Il s'agit, d'une part, de modéliser ces plates-formes en représentant l'organisation des ressources de manière suffisamment simple pour masquer les détails techniques du matériel, et suffisamment précise pour permettre aux algorithmes de prendre des décisions de placement ou d'ordonnancement ; d'autre part, je propose des outils améliorant la gestion des architectures modernes où l'accès à la mémoire et aux périphériques n'est plus uniforme. Cela permet d'améliorer les performances de bibliothèques de calcul parallèle en tenant compte de la localité.

Abstract

Scientific simulation is increasingly involved in industry and high performance computing is therefore important to today's society. However, the more powerful parallel computing platforms become, the more difficult their use is. Indeed, their complexity keeps growing considering the increasing amount of computing and storage resources involved, their heterogeneous features and their non-uniform locality. There is a need for new tools facilitating the exploitation of these parallel platforms.

I first present how specialized high-performance networking mechanisms can be made more accessible and portable. This idea is applied to traditional networks such as Ethernet and communication between local processes, so as to improve message passing (MPI) performance without requiring proprietary hardware technologies.

Then I explain how the management of modern hierarchical computing servers can be eased. Hardware platforms can be modeled based on the resource locality in simple manner, to hide technical details, but precise enough to let placement and scheduling algorithms take relevant decisions. Then I propose several tools enhancing the use of modern architectures where memory or I/O access is non-uniform. They provide performance improvements in parallel computing libraries by taking locality into account.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1
Introduction

Le calcul haute performance (HPC) est désormais partout. Le grand public l'ignore encore souvent, mais la plupart des industries modernes utilise la simulation numérique pour remplacer les expériences lentes et coûteuses. C'est vrai dans l'industrie aéronautique pour concevoir le fuselage des avions, dans l'automobile pour que nos voitures nous protègent mieux des accidents, mais aussi dans bien d'autres comme la pharmaceutique où on simule les interactions moléculaires pour créer les médicaments du futur. Ces nouveaux usagers gourmands en puissance de calcul ont fait considérablement évoluer le HPC, qui n'est plus du tout réservé à quelques communautés de physiciens comme dans le passé.

En parallèle de cette démocratisation, les architectures matérielles ont considérablement évolué. Entre le début de ma thèse et aujourd'hui, plus de dix ans plus tard, l'architecture mémoire des ordinateurs est devenue non uniforme et les processeurs sont devenus multicœurs. Les plus gros supercalculateurs du monde sont passés de quelques milliers de processeurs monocœurs [1] à des dizaines de milliers de machines dont les processeurs, parfois hétérogènes, contiennent une dizaine de cœurs. [2] Si la puissance théorique des processeurs continue à augmenter régulièrement, l'exploiter dans nos programmes est devenu une autre paire de manches [MS4]. Il n'est plus du tout envisageable d'espérer l'exploiter sans tenir compte de cette nouvelle complexité du matériel [MS2].

1.1 Concevoir des outils pour faciliter l'exploitation des supercalculateurs

Le système d'exploitation joue le principal rôle d'intermédiaire entre les applications et le matériel [MS1]. Ce rôle est cependant un peu trop limité pour le HPC, par exemple parce qu'un système d'exploitation classique ne gère qu'un seul ordinateur, et peu de programmes parallèles. On lui adjoint donc une surcouche appelée *support exécutif* pour faire l'articulation entre les milliers de cœurs constituant un supercalculateur, par exemple avec une bibliothèque de communication comme MPI, ou une parallélisation par threads comme OpenMP. Les applications de calcul scientifique peuvent alors utiliser ces interfaces de programmation parallèle directement, ou via des bibliothèques spécialisées pour résoudre certains problèmes, comme par exemple ScaLAPACK pour l'algèbre linéaire. Ces couches sont résumées sur la figure 1.1 dans laquelle je situerai plus loin mes contributions.

Le HPC est actuellement sur la route de l'Exascale. On s'attend à ce qu'une première machine soit capable de soutenir 1 exaflop/s (1 milliard de milliards d'opérations de calcul flottant par seconde) d'ici 2020. La consommation d'énergie est un des principaux obstacles car les technologies actuelles ne peuvent pas atteindre l'exaflop sans nécessiter une tranche de

1. *ASCI Red*, le plus puissant supercalculateur à la fin des années 1990 d'après le classement Top500 [70], était composé de 9 298 processeurs monocœurs.

2. Le plus puissant supercalculateur en 2013, Tianhe-2, contient 16 000 machines contenant deux processeurs à 12 cœurs et 3 accélérateurs à 57 cœurs.

centrale nucléaire dédiée. De nombreuses améliorations seront nécessaires, au niveau matériel ainsi qu'à tous les étages de la pile logicielle.

Si on ignore encore l'architecture des supercalculateurs exaflopiques, la complexité croissante des architectures ces dernières années tend à considérer que les supports exécutifs continueront à avoir un rôle clef. En effet, il leur faudra être capable d'exploiter intelligemment des millions de cœurs organisés de manière très hiérarchique. Et il faudra le faire sans exposer cette complexité de manière trop brutale au programmeur d'applications parallèles, car il ignore souvent tous ces détails matériels (quelle chance !). C'est dans cet axe que se situent mes travaux de recherche.

1.2 Historique de mes travaux de recherche

Mes travaux de thèse se sont placés à l'interface des domaines des réseaux et du stockage. En 2002, les réseaux rapides tels que Myrinet étaient arrivés à maturité pour les communications MPI. Par contre le stockage distribué dans les supercalculateurs continuait à utiliser les réseaux et protocoles traditionnels (IP). Ma thèse, réalisée au sein du Laboratoire de l'Informatique du Parallélisme (LIP) à l'ÉNS Lyon, a consisté à étudier les besoins du stockage distribué afin d'adapter les interfaces de programmation des réseaux rapides [Th1]. Mon travail s'est matérialisé dans l'interface MX des réseaux Myrinet [CI10].

J'ai ensuite poursuivi ces travaux dans l'interface MX au cours d'un séjour post-doctoral dans l'entreprise Myricom dans le Tennessee. Cela a conduit à son utilisation native dans les principaux systèmes de stockage distribué à grand échelle, PVFS et Lustre [RR1]. J'ai également beaucoup travaillé sur l'optimisation des transferts de données dans les serveurs de calcul [RR2] (j'aborderai des aspects similaires dans le chapitre 2), et diagnostiqué les (mauvaises) interactions entre le HPC et les systèmes d'exploitation [RA1] (j'y reviendrai dans la conclusion 6.3.2).

Ce début de carrière m'a permis d'acquérir de vastes connaissances sur les différentes interfaces de programmation et protocoles de communication réseau, que ce soit entre les nœuds ou à l'intérieur [MS3]. Ces travaux se sont concentrés dans les bibliothèques système de bas-niveau et dans les pilotes du système d'exploitation.

Depuis 2004, les processeurs multicœurs et les architectures NUMA se sont généralisés dans les grappes de calcul. Cette évolution matérielle a joué un rôle important dans mes axes de recherche depuis mon arrivée à Inria en 2006 car les communications réseau et les gros serveurs à mémoire partagée ne pouvaient plus être considérés comme des domaines d'étude indépendants.

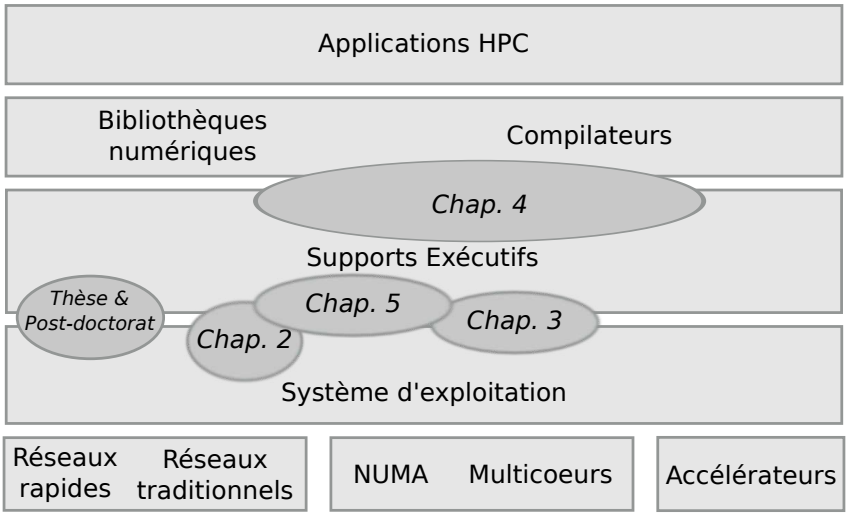


FIGURE 1.1 – Répartition de mes contributions dans les couches logicielles du HPC.

1.3 Organisation du document

Je présente dans ce document mes contributions depuis mon arrivée à Inria en 2006. Elles se répartissent dans 4 chapitres. La figure 1.1 indique où ils se situent dans l'environnement logiciels du HPC.

1.3.1 Rendre portables et accessibles des mécanismes spécialisés des réseaux haute performance

Comme expliqué dans la section précédente, j'ai depuis le début de ma thèse travaillé sur les communications sur réseaux haute performance et leurs pilotes dans le système d'exploitation. J'ai pu constater toutes les innovations logicielles et matérielles que ces technologies apportaient. Après la fin de mon séjour post-doctoral chez Myricom, j'ai souhaité généraliser ces idées pour que tous les centres de calcul en bénéficient, indépendamment de la technologie réseau matérielle. Il s'est donc agi de proposer des innovations similaires accessibles à toutes les applications (sans besoin de les reprogrammer) et utilisables sur toutes les plates-formes. Cet axe de recherche se décline en deux chapitres :

— Le chapitre 2 s'intéresse aux communications inter-nœud. Il présente mon travail autour du logiciel Open-MX qui visait à proposer une implémentation efficace du standard de communication MPI au dessus de technologies classiques Ethernet.

— Le chapitre 3 cible quant à lui les communications MPI intra-nœud avec le logiciel KNEM. J'y montre comment proposer des transferts performants entre tâches s'exécutant sur la même machine, sans dépendre de pilotes réseau spécialisés, et suffisamment génériques pour s'appliquer à de nombreux besoins (communications point-à-point, collectives, etc.).

1.3.2 Faciliter l'exploitation des calculateurs hiérarchiques modernes

La démocratisation des architectures NUMA et multicœurs dans le HPC depuis ma thèse m'a conduit à travailler à la simplification de l'exploitation de ces machines plus complexes. Ce second axe de recherche se situe à l'intersection des communications dans les grappes et du calcul sur gros serveurs SMP. Il se décline également en deux chapitres :

— Le chapitre 4 présente mes idées pour modéliser la complexité de ces architectures modernes. Il s'agit d'une part de représenter la hiérarchie matérielle et les affinités entre les différentes ressources de manière portable avec le logiciel hwloc. D'autre part, on tente de comprendre l'influence de la hiérarchie mémoire puis en extraire des critères quantitatifs pouvant aider à la prise de décision selon les affinités matérielles et logicielles.

— Le chapitre 5 s'intéresse quant à lui spécifiquement aux architectures NUMA qui apportent de nombreux problèmes liés au placement et à la migration mémoire. Il s'agit d'une part d'être capable de correctement gérer les affinités entre les tâches, les caches et la mémoire, et d'autre part d'étudier l'impact de cette localité sur les performances d'accès aux périphériques d'entrées-sorties.

En conclusion 6, je discuterai d'aspects importants pour la recherche dans ces domaines, en particulier comment appréhender le développement logiciel pour nos travaux, la diffusion de ces résultats, et la cohabitation avec les communautés proches. Puis je proposerai quelques perspectives pour les prochaines années.

Chapitre 2

Passage de messages haute performance sur réseau Ethernet standard

Les communications entre les machines de calcul sont un élément critique de la performance globale du système. Un accès à une machine distante est en effet beaucoup plus lent qu'un accès local. Le débit réseau dans une grappe de calcul reste limité à quelques gigaoctets par seconde et la latence est souvent de quelques microsecondes, soit 1 ou 2 ordres de grandeur plus lent que les débits et latences à l'intérieur d'une machine.

On dit souvent que les entrées-sorties sont le *maillon faible* d'un ordinateur car tous les périphériques vont beaucoup plus lentement que le *cœur* de la machine, les processeurs et la mémoire. C'est particulièrement vrai pour les communications réseau dans les applications parallèles comme je l'explique dans l'article de vulgarisation scientifique [MS3]. Le moindre ralentissement réseau peut mettre certains cœurs au chômage technique par manque de données à traiter. Les étapes de synchronisations peuvent ensuite propager ce retard aux milliers d'autres cœurs de la plate-forme de calcul. Le coût de fonctionnement de ces plates-formes interdisant un tel gaspillage de temps et d'argent, les communications réseau sont donc naturellement un sujet récurrent de recherche.

Depuis une vingtaine d'années, on a assisté à de nombreuses innovations logicielles et matérielles dans ce domaine. Les réseaux *rapides* du calcul parallèle se répartissent depuis entre des technologies matérielles dédiées (Myrinet, QsNet, InfiniBand...) poussées par les industriels, et des technologies plus standards (Ethernet et TCP/IP). Ma carrière m'a amené à beaucoup travailler sur ces aspects, du côté académique mais aussi du côté industriel pendant mon post-doctorat chez Myricom. Une des idées marquantes de ces dix dernières années a consisté en la convergence *prochaine* de tous ces travaux vers une technologie ultime qui serait à la fois standard et haute performance et servirait à la fois au calcul parallèle et à monsieur tout le monde.

Un de mes axes principaux de travail depuis mon arrivée à Inria a été d'étudier cette convergence et notamment tenter de savoir quelles technologies matérielles et logicielles en résulteraient. Il s'est notamment agi de transposer les idées des réseaux rapides spécialisés vers les réseaux traditionnels, dans la lignée de mon premier grand axe de recherche présenté en section 1.3.1.

J'ai notamment mené ces travaux dans le logiciel Open-MX qui est né d'une collaboration contractuelle entre Inria et Myricom en 2007. Je rappelle dans ce chapitre les grands principes des communications dans les applications parallèles et les technologies réseau qui en découlent. Je montre ensuite comment mes travaux dans Open-MX puis CCI ont contribué à proposer une solution aux problèmes des piles réseau logicielles standards dans le contexte du calcul haute performance sur technologie réseau générique.

2.1 Il était une fois des réseaux spécialisés pour le HPC

La démocratisation des grappes de calcul (*clusters*) est née des projets NOW [3] et Beowulf [63] au début des années 1990. Ils ont proposé d'assembler des dizaines de stations de travail *normales* en grandes plates-formes de calcul. Cette idée présente l'avantage indéniable d'utiliser des technologies traditionnelles et donc un coût de revient très intéressant.

Cependant, les réseaux traditionnels qui reliaient alors ces machines ont rapidement fait l'objet de nombreux travaux de recherche afin de réduire l'impact de ces communications externes sur la puissance globale du système. De nombreuses innovations, matérielles et logicielles, ont depuis été proposées pour ce contexte très particulier où le standard de communication MPI est roi.

2.1.1 Innovations technologiques au détriment de la standardisation

Les grappes de calcul ont conquis le marché du calcul haute performance à la fin des années 1990 [70]. Si les supercalculateurs les plus puissants ont longtemps continué à utiliser des technologies dédiées, très onéreuses, la plupart des centres de calcul ont progressivement adopté l'architecture des grappes qui reposent sur du matériel quasiment standard et donc un meilleur rapport qualité/prix. Leurs performances étant plus limitées, de nombreuses technologies réseau ont vu le jour afin de réduire l'écart en optimisant le chemin critique entre les différentes machines.

Parmi les principales, on peut notamment citer Myrinet [11] de Myricom, qui a occupé entre 10 et 30% du Top500 [70] de 1999 à 2007, et a forgé son succès sur des cartes réseau implémentant l'interface MPI de manière beaucoup plus rapide que les réseaux traditionnels. Quadrics QsNet [57], souvent considéré comme la *Rolls* des réseaux rapides, atteignait même des latences de l'ordre de la microseconde, encore jalousées 10 ans plus tard. Par ailleurs, de nombreux développements logiciels ont été menés pour mieux tirer parti de ces technologies matérielles pour le calcul haute performance, par exemple en proposant de nouvelles interfaces de programmation réseau, telles que *Fast Messages* [54].

De telles technologies ont fait progresser significativement le support logiciel et matériel pour les communications dans les calculateurs distribués. Elles ont dominé le calcul haute performance pendant une dizaine d'années mais sont tout de même restées limitées à ce marché de niche. Par ailleurs ce marché a longtemps eu la particularité d'avoir pour principaux utilisateurs des chercheurs, en particulier dans les autres sciences. Cette communauté a des besoins très particuliers, la performance étant notamment plus importante que la fiabilité du système ou sa standardisation. Des innovations très diverses ont donc pu être proposées, que ce soit dans le matériel ou dans le support logiciel, du système d'exploitation aux interfaces de programmation, au détriment de l'uniformisation des solutions.

Les constructeurs ont notamment pu revoir toute l'organisation matérielle du réseau pour en changer la topologie, les cartes d'interface, les protocoles de routage et de gestion de la congestion. En effet, les grappes sont un environnement beaucoup plus maîtrisé qu'un réseau normal puisque les utilisateurs et applications sont connus et les interconnexions avec l'extérieur souvent limitées. Le cœur de réseau peut donc être dimensionné correctement et les protocoles adaptés pour tenir compte des faibles distances, de la topologie régulière (routage par la source), du peu de congestion, et des rares pertes de paquets. Au niveau des machines, cela a permis d'optimiser significativement les performances en évitant notamment les épaisses couches assurant la gestion des réseaux non maîtrisés (par exemple TCP/IP).

Ces modifications en profondeur de la façon d'utiliser le matériel réseau se sont accompagnées de profondes modifications des couches logicielles. En effet, le système d'exploitation et

les pilotes sont utilisés le moins possible afin de réduire le chemin critique (*OS-Bypass*, *Zéro-copie*, etc.). Les applications disposent d'un accès privilégié et direct à la carte réseau [1], ce qui impose des contraintes sur le nombre de tâches et leur comportement. Ces fonctionnalités particulières ont conduit les pilotes de réseaux rapides à utiliser des implémentations significativement différentes des systèmes d'exploitation habituels. Quadrics a par exemple rapidement recouru à des modifications du noyau Linux pour améliorer les performances des transferts DMA sans nécessiter de désactiver certains mécanismes de la mémoire virtuelle (le *Swap*). Le calcul haute performance a donc été très mal considéré par la communauté du noyau Linux au début des années 2000 car il apparaissait comme un domaine ne cherchant que la performance extrême au détriment de tous les standards logiciels [2]. Ce n'était pas complètement faux, mais cela montrait aussi une forte incompréhension des besoins du calcul haute performance par la communauté Linux. Heureusement, cela s'est progressivement amélioré depuis.

2.1.2 InfiniBand, enfin un standard... pour un marché de niche

Le début des années 2000 a vu une révolution lentement se mettre en place avec la définition du standard InfiniBand [35] par un consortium regroupant tous les grands industriels. L'objectif initial était de concevoir une technologie remplaçant aussi bien les bus d'entrées-sorties des machines et l'interconnexion entre elles et avec le stockage. Intel s'étant finalement retiré pour créer *PCI Express* (PCIe), l'initiative s'est recentrée autour des réseaux rapides pour le HPC (avant de revenir plus récemment vers le stockage).

Les technologies logicielles et matérielles sous-jacentes utilisées dans InfiniBand n'étaient pas révolutionnaires. Elles reprenaient très largement les idées des technologies précédentes telles que QsNet ou Myrinet. Ce travail s'est matérialisé sous la forme de standard de communication autour du RDMA (*Remote Direct Memory Access*) qui définit le transfert de données entre les mémoires de tâches et machines distantes. Ce modèle permet d'implémenter du passage de messages MPI relativement aisément [3].

Devant l'importance croissante du calcul haute performance dans la société moderne, le consortium a veillé à sortir de l'amateurisme existant en définissant des spécifications et standards officiels pour cette interface de communication et les protocoles réseau associés... pouvant ainsi être adoptés par une communauté beaucoup plus vaste et nécessiteuse. Cette initiative a commencé à porter ses fruits vers 2003 quand la technologie InfiniBand a pu rivaliser en terme de performance avec Myrinet ou Quadrics [46]. Elle s'est réellement généralisée, par exemple dans le Top500, depuis 2007 et y représente désormais environ 40%, autant que les technologies Ethernet traditionnelles.

InfiniBand a eu l'énorme avantage d'apporter enfin un standard dans la jungle des technologies logicielles pour le HPC. La pile logicielle a été incluse dans le noyau Linux officiel en 2005, en y apportant un lot de fonctionnalités longtemps désirées par la communauté du HPC. Le verrouillage des pages pour le DMA entre la mémoire des applications et la carte réseau a ainsi été grandement facilité, permettant des communications zéro-copie aisées. Il est devenu possible d'être notifié des modifications d'espace d'adressage virtuel (*MMU Notifiers*), facilitant encore plus le zéro-copie, alors que cette idée avait été décriée précédemment [4]. La

1. La mémoire de la carte réseau est projetée dans la mémoire virtuelle de l'application.
2. En 2006, alors que je travaillais sur le pilote MX de Myrinet pendant mon séjour post-doctoral, Arnd Bergmann, développeur noyau et ingénieur IBM, m'avait confessé devoir recommander à ses clients de ne pas utiliser les réseaux rapides car le code de leurs pilotes était trop *horrible*.
3. Le constructeur Mellanox, leader du marché InfiniBand, a cependant récemment introduit l'interface alternative *MellanoX Messaging* (MXM) censée fournir une implémentation MPI plus efficace sur son matériel InfiniBand. On y reviendra en section 2.4.3.
4. `http://lkml.org/lkml/2005/4/29/175` montre que Roland Dreier, mainteneur d'InfiniBand dans le

projection de mémoire des périphériques en mémoire virtuelle a plus récemment vu l'ajout d'options de configuration du cache (*Write-Combining*) si longtemps désirées pour simplifier l'obtention de latences réseau faibles [5].

Ces innovations ont contribué à tout l'écosystème du calcul haute performance. D'une part elles ont permis aux autres technologies d'améliorer leur support logiciel en en bénéficiant. D'autre part, elles ont fait évoluer l'image du groupe de hackers avides de performance vers celle d'une communauté sérieuse et développant des standards.

Même s'il s'est diffusé dans la société pour répondre aux besoins croissants en simulation numérique, le calcul haute performance reste cependant un domaine très spécifique. Les réseaux tels que InfiniBand sont encore cantonnés à un marché de niche, malgré des tentatives d'incursions dans le domaine du stockage ou des datacenters pour des raisons de débits réseau importants. La technologie n'est d'ailleurs produite que par deux constructeurs, Mellanox et Intel [6].

2.1.3 L'hypothétique convergence d'Ethernet et des réseaux rapides

L'émergence d'InfiniBand comme standard de communication dans les grappes de calcul s'est accompagnée de l'annonce récurrente de l'imminence d'une convergence des différents types de réseaux. Cette idée s'est concrétisée dans le contexte du stockage où la technologie *FibreChannel-over-Ethernet* (FCoE [24]) semble désormais le standard d'avenir pour l'accès au stockage distant dans les datacenters. InfiniBand tente également une incursion dans le monde du stockage, comme prévu à l'origine du consortium, en mettant en avant les grands débits en jeu dans ce domaine, mais son adoption reste encore limitée [7]. En effet, l'importance du marché du stockage, beaucoup plus vaste que celui du HPC, a attiré de nombreux constructeurs.

Depuis 2005, on annonce l'imminence de la fusion de tous ces réseaux, traditionnels (Ethernet), stockage (FibreChannel) et réseaux rapides pour le HPC. La raison derrière cette mode est tout d'abord la nécessité pour la niche du HPC de s'ouvrir aux autres communautés. En contrepartie, les avancées technologiques du HPC, notamment en terme de performances, pourraient bénéficier à bien d'autres domaines que les communications MPI dans les applications parallèles.

Si la convergence annoncée a fait l'objet d'innovations concrètes, force est de constater que 7 ans n'ont pas suffi à l'imposer. Une des raisons à cela est qu'elle s'est souvent limitée à quelques couches de la pile réseau sans jamais se généraliser de haut en bas. La plupart des réseaux modernes peuvent par exemple utiliser des câblages similaires, mais le signal utilisé par InfiniBand est différent de celui d'Ethernet. Dans le cas de Myricom, c'est le protocole de routage qui est différent.

Plus récemment, InfiniBand a aussi promu une certaine convergence logicielle en tentant d'appliquer son modèle RDMA aux autres technologies réseau. La première tentative a conduit à la création de iWarp, qui utilise des cartes réseau spéciales pour permettre du RDMA dans les connexions TCP [59] mais l'intérêt est limité aux connexions TCP longue distance. Enfin, Mellanox a lancé RoCEE (*RDMA over Converged Enhanced Ethernet*) qui propose le protocole InfiniBand directement sur la couche Ethernet. Les performances sont cette fois au rendez-vous et montrent que les couches basses d'Ethernet n'ont rien à envier à InfiniBand. Cela contredit

noyau Linux, n'avait pas encore compris l'intérêt de cette fonctionnalité quand Quadrics l'a présentée 3 ans plus tôt.

5. `http://lkml.org/lkml/2007/8/8/145` est un des nombreux échanges bloquant cette fonctionnalité par manque de généricité de la solution proposée.

6. Intel a racheté la branche InfiniBand de QLogic en 2012.

7. L'adaptation de NFS/RDMA du système de fichiers distribués le plus répandu, NFS, sur l'interface RDMA d'InfiniBand n'est par exemple toujours pas considérée comme stable.

bien la croyance répandue que le HPC boude les réseaux traditionnels à cause des lacunes d'Ethernet. Le surcoût inadapté au calcul parallèle est en fait dans TCP/IP. D'ailleurs, dans le stockage, cela s'est traduit par les technologies FCoE et AoE (*ATA-over-Ethernet* [4]) qui court-circuitent les couches TCP/IP pour directement implémenter leur protocole sur réseau local Ethernet.

Avec plusieurs années de recul, toutes ces propositions laissent penser que la convergence annoncée entre Ethernet et les réseaux rapides est essentiellement un argument marketing visant à convaincre que les réseaux du HPC sont compatibles et intéressants pour d'autres domaines. En pratique, si certaines couches sont interchangeables, les deux mondes restent assez largement incompatibles. Les réseaux rapides continuent donc à apparaître comme une technologie pour un marché de niche où la pile réseau est très spécifique. L'intégration promise de puces InfiniBand sur les cartes mères ne se fait que dans les serveurs dédiés au HPC dense. L'hégémonie de Ethernet auprès du grand public impose d'être compatible avec Ethernet pour espérer y percer. On parle d'ailleurs souvent de *Ethernet vs EtherNOT* pour expliquer que les concurrents d'Ethernet ne pourront jamais s'imposer largement car ils ne sont pas l'Ethernet que le grand public et les investisseurs souhaitent.

2.2 Faire du MPI avec les réseaux standards

Malgré le succès des réseaux rapides dans le domaine du HPC, leur coût conduit les centres de calcul à n'y investir que quand les performances des communications leur sont critiques. Beaucoup d'applications parallèles et d'utilisateurs peuvent en effet se satisfaire de communications un peu plus lentes. Les réseaux traditionnels Ethernet sont donc très utilisés pour le calcul parallèle. Ils représentent d'ailleurs 40% des 500 plus grands calculateurs du monde [70]. S'il doit y avoir convergence avec les réseaux rapides, la question est de savoir quelles fonctionnalités matérielles et logicielles en résulteront. En attendant, améliorer les communications MPI sur Ethernet sert dans un très grand nombre de grappes.

Mes travaux se placent dans l'intervalle de performance entre les réseaux traditionnels et les réseaux rapides. L'objectif est de marier les cartes Ethernet modernes, qui disposent de fonctionnalités avancées, mais plutôt spécifiques à TCP/IP, avec les interface de programmation des réseaux rapides, afin de bénéficier à toutes les implémentations MPI répandues. Ce travail a été mené dans le logiciel Open-MX [RI2] [8], développé dans le cadre d'une collaboration contractuelle avec Myricom. Il s'inscrit dans le contexte de la convergence des réseaux puisqu'il vise à implémenter la pile protocolaire MX de Myricom sur des réseaux Ethernet traditionnels.

2.2.1 Ethernet n'est pas obsolète, ni lent

Ethernet est souvent dit lent à cause de la confusion avec la pile TCP/IP. Cette dernière est considérée depuis longtemps comme responsable des performances limitées de MPI sur Ethernet [9], notamment à cause des copies mémoire [20]. Le protocole TCP ne peut cependant pas être modifié en profondeur sans casser sa compatibilité avec les installations existantes. Des protocoles concurrents dédiés aux grappes ont donc été proposés, par exemple BIC (*Binary Increase Congestion control*). Mais quitte à modifier TCP, autant modifier toutes les couches protocolaires qui ne sont pas adaptées aux grappes de calcul, et donc notamment IP car le routage importe peu dans le réseau local d'un centre de calcul.

Là où TCP/IP impose beaucoup de choses sur le protocole et l'interface de communication, Ethernet laisse l'utilisateur très libre et permet d'envisager des communications très perfor-

8. Open-MX est disponible sous licence GPL/LGPL à `http://open-mx.gforge.inria.fr`.

mantes, comme on l'a vu avec FCoE et AoE. Au lieu de dire qu'Ethernet est moins performant qu'InfiniBand, il faut en fait bien comprendre que c'est TCP/IP sur Ethernet qui n'est pas au niveau de l'interface RDMA sur InfiniBand pour les besoins du HPC. D'ailleurs TCP/IP n'est pas réputé pour fonctionner très bien sur InfiniBand non plus.

Des recherches ont été menées pour concevoir des piles protocolaires dédiées au HPC en évitant l'intégralité de la pile TCP/IP. Cependant, la plupart d'entre elles nécessitait des modifications matérielles ou logicielles, ou n'atteignaient pas les grands débits ou faibles latences promises par les cartes Ethernet 10G. On peut notamment citer GAMMA [19], PM-Ethernet [64] ou MultiEdge [42] qui modifient les pilotes de manière intrusive, ou EMP [60] qui utilise un micro-programme dédié dans la carte réseau. De plus, ces projets proposent une interface de programmation et une implémentation MPI spécifique au lieu de les intégrer dans des implémentations populaires comme MPICH ou Open MPI qui sont considérées comme stables, complètes et performantes.

Ces dernières années, de nombreuses innovations ont été apportées aux cartes réseau Ethernet afin d'améliorer les performances de TCP/IP. Elles peuvent désormais segmenter les paquets en émission (TSO, *Transmit Segmentation Offload*), calculer les sommes de contrôle, ou répartir intelligemment le traitement des différents flux entre les différents cœurs de la machine (*Multiqueue* [77]). En parallèle, des innovations logicielles permettent maintenant de réassembler les paquets TCP en réception avant leur traitement (LRO, *Large Receive Offload* [30]) ou de voler des connexions entre cœurs afin de traiter les paquets reçus près de l'application cible (RFS, *Receive Flow Steering* [22]). Ces travaux montrent que des techniques avancées peuvent être appliquées au modèle simple d'Ethernet pour améliorer les performances. Mais elles sont spécialisées pour les communications par flux, typiquement TCP/IP, et pas pour le passage de messages à la MPI.

Il faut noter que, parmi tous les standards réseau impliqués dans le HPC, Ethernet est un de ceux qui évoluent le plus. Les protocoles TCP/IP peuvent difficilement évoluer sans casser la compatibilité avec les installations existantes. InfiniBand n'évolue guère qu'en augmentant son débit [9] et en réduisant sa latence [10]. Ethernet connaît également des évolutions au niveau des débits (100 Gbit/s a été standardisé récemment) mais de nouvelles fonctionnalités ont également été récemment ajoutées. Par exemple CEE (*Converged Enhanced Ethernet*) qui permet de traiter matériellement différents trafics et de leur fournir des garanties différentes [11]. Cette idée s'applique directement à la convergence annoncée des différents réseaux puisque le stockage, les communications MPI et les communications traditionnelles n'ont pas les mêmes besoins.

2.2.2 Remplacer la puissance de la NIC par celle du processeur central

Comme une très grande part des grappes n'utilisent que des cartes réseau traditionnelles, il est important de fournir des implémentations MPI fonctionnant efficacement sans requérir de fonctionnalités matérielles avancées dont seuls les réseaux rapides disposent. À partir du moment où la carte réseau ne peut pas prendre en charge une grande partie du traitement réseau, il doit être rapatrié dans l'hôte, c'est-à-dire sur le processeur central (on parle d'*Onload* par opposition à l'*Offload*).

De même, comme les cartes ne sont pas capables d'exposer leurs fonctionnalités directement aux applications, le système d'exploitation doit être utilisé pour en coordonner l'utilisation. Comme illustré sur la figure 2.1, notre modèle doit donc abandonner l'*OS-Bypass* cher aux développeurs des premiers réseaux rapides pour améliorer la latence. Heureusement, les

9. FDR, *Fourteen Data Rate*, 56 Gbit/s de signal, vient d'apparaître et EDR est déjà annoncé à 100 Gbit/s.

10. La réduction de latence est désormais essentiellement liée à l'amélioration des processeurs et des bus d'entrées-sorties.

11. CEE est aussi parfois appelé DCE (*Data Center Ethernet*).

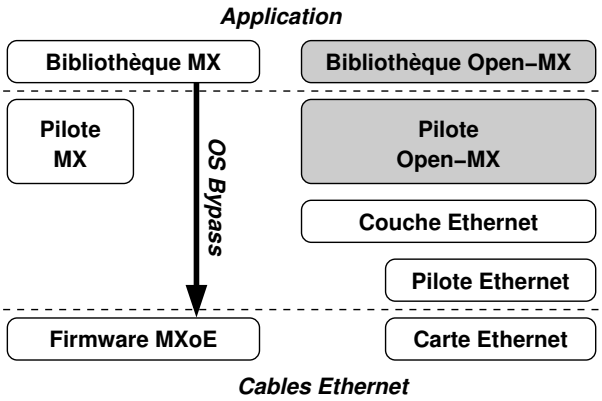


FIGURE 2.1 – Comparaison des piles réseau MXoE et Open-MX. Open-MX utilise les couches Ethernet logicielles et matérielles standards. MXoE (*MX-over-Ethernet*) accède directement à la carte Myri-10G et à son micro-programme spécialisé pour le passage de messages et configuré pour Ethernet.

contraintes temporelles ont évolué. Le coût d'un appel-système est inférieur à 100 nanosecondes depuis plusieurs années, ce qui est très largement acceptable quand la latence réseau est de l'ordre de quelques microsecondes [RR2].

L'absence d'intelligence dans la carte réseau interdit de plus aux données entrantes d'être directement déposées dans la mémoire de l'application destinataire. Ce modèle impose en effet de prévenir préalablement la carte que tel message doit être reçu dans telle zone mémoire, ce qui est très similaire aux fonctions de *Matching* ou RDMA des réseaux rapides. En leur absence, la carte dépose les données dans un anneau de réception et le système d'exploitation va ensuite les passer à l'application destinataire, ce qui implique une copie mémoire supplémentaire, comme montré par la table 2.1.

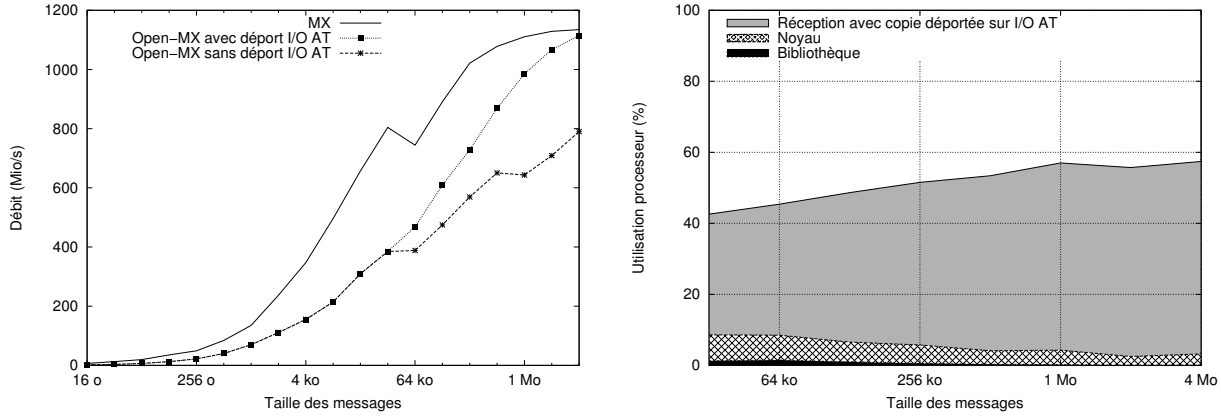
Taille des	Open-MX		MX	
messages	Noyau	Bibliothèque	Carte réseau	Bibliothèque
jusqu'à 64 ko	Copie	Copie	DMA	Copie
au-delà	Copie	Verrouillage	DMA	Verrouillage

TABLE 2.1 – Les stratégies de réception dans Open-MX et MX selon la taille des messages. Open-MX nécessite une copie additionnelle à cause de l'absence de support matériel.

Si l'absence de fonctionnalités avancées dans les cartes réseau impose de nombreuses contraintes, leur déport dans le système d'exploitation apporte lui un certain nombre d'avantages. En particulier, le fait de centraliser tout le traitement à un seul endroit plutôt que de le répartir entre l'hôte et la carte réseau réduit les besoins en synchronisation. En effet, les communications zéro-copie imposent par exemple d'invalider les descripteurs de zones mémoires dans la carte quand le système d'exploitation modifie l'espace virtuel d'une application. Cela impose des étapes d'initialisation synchrones coûteuses pour verrouiller les pages virtuelles en mémoire physique. Dans notre modèle, tout étant géré dans le système d'exploitation, les interactions entre la mémoire virtuelle et les communications sont très simples. J'ai par exemple montré qu'on pouvait complètement cacher et recouvrir ces étapes d'initialisation en découplant le comportement de l'application d'une part, et la gestion des communications et de la mémoire virtuelle d'autre part [WI2].

2.2.3 Copies mémoire chères... ou pas

L'ajout d'une copie mémoire sur le chemin en réception est un obstacle sévère à l'obtention de performances élevées pour le passage de messages sur Ethernet, en particulier sur les architectures à bus mémoire centralisé qui forme rapidement un goulet d'étranglement. J'ai montré que ces copies mémoires pouvaient être efficacement déportées sur les moteurs I/O AT DMA qu'embarquent les processeurs modernes. [12] Cette idée s'applique très bien au modèle du passage de messages car elle permet de traiter en tâche de fond de gros ensembles de paquets réseau sans occuper le processeur ou polluer les caches. Ce travail m'a permis de contourner les limites de débit en réception et d'atteindre 10 Gbit/s comme montré sur la figure 2.2 et détaillé dans [CI5].



(a) Débit (b) Utilisation processeur

FIGURE 2.2 – Intérêt du déport des copies en réception sur la DMA Engine des plates-formes Intel I/OAT. Open-MX atteint ainsi 10 Gbit/s tout en n'utilisant que la moitié du temps disponible sur un cœur de processeur Intel Xeon E5345 cadencé à 2,33 GHz.

Ces travaux ont perdu de leur importance ces dernières années car la révolution des bus mémoire des architectures Intel a changé la donne. [13] En effet, la bande passante des copies mémoire opérées par le processeur a été démultipliée tandis que les composants I/O AT évoluaient peu. Cependant, la situation pourrait redevenir favorable aux copies déportées car l'intégration du contrôleur d'entrées-sorties directement dans les processeurs Intel *Sandy-Bridge* améliore significativement ses performances.

Un moyen d'éviter les copies mémoire consiste à utiliser la mémoire virtuelle pour déplacer virtuellement une zone de mémoire physique. Cette idée a été étudiée depuis longtemps dans le contexte des réseaux [18]. Son principal inconvénient est d'imposer l'alignement en mémoire virtuelle des zones mémoires manipulées. Il faut une forte collaboration des pilotes et des protocoles réseau pour le contourner [56]. L'autre inconvénient de cette technique est qu'elle impose de complexes manipulations de la table de pages qui sont chères sur les machines multicœurs modernes.

Au final, de nombreux développeurs considèrent qu'il vaut mieux accepter d'avoir à faire des copies mémoire car celles ci ne sont plus si chères sur les processeurs modernes. Cepen-

12. Intel I/O Acceleration Technology (I/O AT) embarque dans les puces un *DMA Engine* capable d'effectuer des copies mémoire en arrière plan.

13. La micro-architecture Intel *Nehalem* a introduit une interconnexion mémoire NUMA où chaque processeur dispose de ses bancs mémoire locaux, par opposition au vieux bus mémoire centralisé.

dant, avec l'annonce de l'arrivée du *Memory Wall* [14], ce constat pourrait avoir à évoluer, et les manipulations de mémoire virtuelle pourraient revenir sur le devant de la scène.

2.3 Profiter du matériel sans en dépendre

J'ai développé Open-MX avec l'idée qu'il n'impose pas la disponibilité de matériel réseau dédié. Cependant, cette portabilité n'interdit pas de bénéficier d'éventuelles fonctionnalités matérielles avancées quand elles sont disponibles. Ce qu'il faut, c'est ne pas concevoir le logiciel en présupposant que le matériel est intelligent. Ce mode de pensée est directement inspiré d'une vieille dispute entre les partisans et les détracteurs du déport de fonctionnalités logicielles dans le matériel. La question est désormais de savoir quel type de fonctionnalité il est une bonne idée de déporter.

2.3.1 Stateful offload décrié, stateless offload à la rescousse

Une des questions à se poser lors du déport de fonctionnalités dans une carte réseau est : quelle est la quantité d'*état* que la carte devra gérer pour assurer cette fonctionnalité ? On parle de *Stateless Offload* (aucun état) ou *Stateful Offload* (déport avec état). Segmenter un paquet en émission ne demande aucun état (le paquet est auto-suffisant). Réassembler en réception demande de conserver les paquets précédemment reçus dans chaque connexion, et donc beaucoup plus de mémoire et de ressources de calcul. Quant au RDMA, il impose de stocker la description des fenêtres mémoire et de maintenir les adresses physiques correspondantes à jour vis à vis des tables de pages dans le système d'exploitation.

Maintenir un état dans la carte réseau à jour vis-à-vis du système d'exploitation de l'hôte impose un mécanisme de synchronisation explicite entre les deux entités. Dans le cas du RDMA, cette synchronisation ne concerne que l'emplacement mémoire des données, qui varie peu. Son coût reste cependant élevé et a amené de nombreux travaux logiciels pour le réduire ou le masquer, en particulier le cache d'enregistrement mémoire [65]. La solution n'est cependant toujours pas idéale car la synchronisation n'est pas seulement entre carte réseau et système d'exploitation, mais aussi avec l'application en espace utilisateur. En effet, la plupart des interfaces de programmation laissent l'application intégralement gérer l'enregistrement alors qu'il est fortement contraint par le système de gestion de la mémoire virtuelle. J'ai montré qu'on pouvait en fait concentrer cette gestion de manière transparente dans les pilotes afin de simplifier son interaction avec le système d'exploitation en évitant de repasser inutilement par l'application [WI2].

Si l'état à maintenir dans la carte évolue au cours du trafic, les besoins en synchronisation sont beaucoup plus importants. C'est pour cette raison que les techniques visant à déporter l'intégralité de la pile TCP dans la carte réseau (*TCP Offload Engine*) ont eu peu de succès. Si l'idée est jolie, la collaboration avec le système d'exploitation est difficile à cause des nombreuses fonctionnalités réseau impliquées, par exemple un pare-feu ou une passerelle, qui doivent connaître l'état maintenu par la carte. [15]

Devant les problèmes du *Stateful*, la tendance est désormais au *Stateless Offload* qui se base des fonctionnalités simples sans état dans la carte pour améliorer les performances. Cette idée est notamment mise en œuvre pour la segmentation des paquets en émission ou la répartition

14. Les performances de la mémoire évoluent moins rapidement que celles des processeurs et que le nombre de cœurs, ce qui conduit à envisager un goulet d'étranglement lors de l'accès aux données dans les prochaines années. On y reviendra en conclusion 6.4.3.

15. TOE est banni du noyau Linux comme expliqué à `http://www.linuxfoundation.org/collaborate/workgroups/networking/toe`.

des paquets dans différentes files de réception (*Multiqueue* [77]). Nous avons étudié cette piste dans le contexte du calcul haute performance.

2.3.2 Affinité en réception avec multiqueue intelligent

La démocratisation des processeurs multicœurs a décuplé les problèmes d'affinité de cache. Il devient désormais critique de veiller à ce que les données ne soient pas manipulées par de multiples cœurs potentiellement très éloignés. Dans le cadre du traitement des paquets réseau, cela consiste notamment à faire en sorte que les différentes étapes du traitement soient effectuées autant que possible par un même cœur. Ainsi les données manipulées restent dans les caches proches de ce cœur au lieu de faire des aller-retours entre différents caches de la machine.

Je me suis attaqué à ce problème dans le contexte des communication MPI sur réseau Ethernet. Le côté émetteur ne présente pas de problème puisque les paquets sont (en général) traités immédiatement par le pilote du système d'exploitation lors de la requête d'émission de l'application, donc sur le même cœur. Par contre, côté récepteur, le traitant du système d'exploitation qui va recevoir un paquet ne sait pas d'avance à quel processus il est destiné, et donc ne sait pas quel cœur va avoir besoin des données.

J'ai résolu ce problème en utilisant la fonctionnalité *Multiqueue* [16] des cartes Ethernet modernes. Cette fonction a été conçue pour assurer que la partie noyau du traitement en réception d'un flux est toujours effectuée sur le même cœur. Il s'agit de diviser la file de réception de la carte réseau en une file par cœur. La carte décide alors du cœur à qui confier un paquet en regardant la connexion TCP dont il fait partie. Cette fonctionnalité n'utilise aucun état dans la carte réseau. Il suffit simplement d'appliquer une fonction de hachage au quadruplet constitué des adresses IP et ports sources et destinations, ce qui ne nécessite que peu de ressources de calcul.

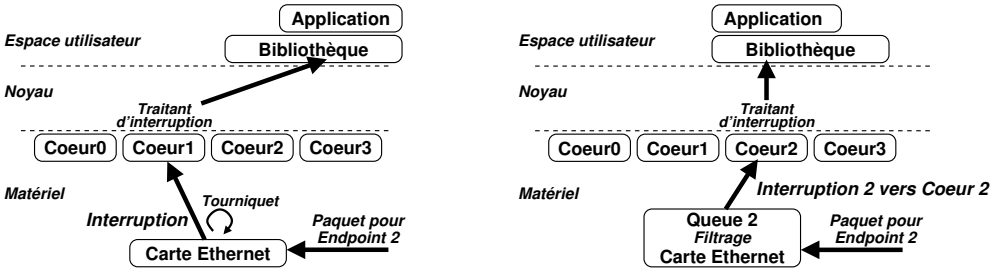
J'ai détourné cette fonctionnalité pour non seulement regrouper le traitement des paquets destinés à un même processus, mais en plus faire en sorte que ce traitement soit sur le cœur où s'exécute ce processus. Il a tout d'abord fallu aider la carte réseau à trier les paquets Open-MX selon le processus destinataire. Comme chaque processus est identifié par l'*endpoint* qu'il a ouvert, il suffit de décider du cœur destination selon le numéro d'endpoint affiché dans l'entête du paquet (une dizaine de ligne dans le micro-programme de la carte réseau).

Il a ensuite fallu choisir entre déplacer le traitement noyau près du processus cible, ou le contraire. Le traitement dans le noyau étant contrôlé par la carte réseau et son pilote, il est préférable de ne pas y toucher. Placer le processus près du traitement réseau dans le noyau est facile dans le contexte du HPC car il y a généralement un seul processus par cœur. Il me suffit donc d'influer sur la décision de placement des processus lors du démarrage de l'application parallèle pour améliorer ses performances réseau. Ce modèle est résumé dans la figure 2.3.

Ces travaux ont été détaillés dans [RI3] et un exemple de gain est présenté sur la figure 2.4. Ils ont permis d'améliorer significativement l'affinité pour le cache de la pile de réception d'Open-MX. Un micro-benchmark configuré finement à la main pourra avoir de meilleures performances que ma solution. Mais pour des applications réelles, où le schéma de communication est souvent mal connu ou trop compliqué pour permettre un placement optimal à la main, ma solution fournit des améliorations immédiates.

Depuis mes travaux, d'autres innovations assez similaires ont été proposées pour les flux TCP. Par exemple le *Receive Flow Steering* [17] permet de forcer le traitement des paquets sur le cœur où se trouve le processus destination. Cette technique est plutôt destinée aux machines

16. Aussi appelé RSS pour *Receive Side Scaling*.
17. http://lwn.net/Articles/382428/.



(a) Par défaut, le traitement de l'interruption et l'application sont exécutés sur des cœurs quelconque.

(b) Quand les interruptions sont dirigées selon l'affinité des paquets Open-MX, leur traitement est effectué sur le cœur où s'exécute le processus destination.

FIGURE 2.3 – Traitement de la réception d'un paquet dans Open-MX avec et sans traitement des interruptions selon le processus cible.

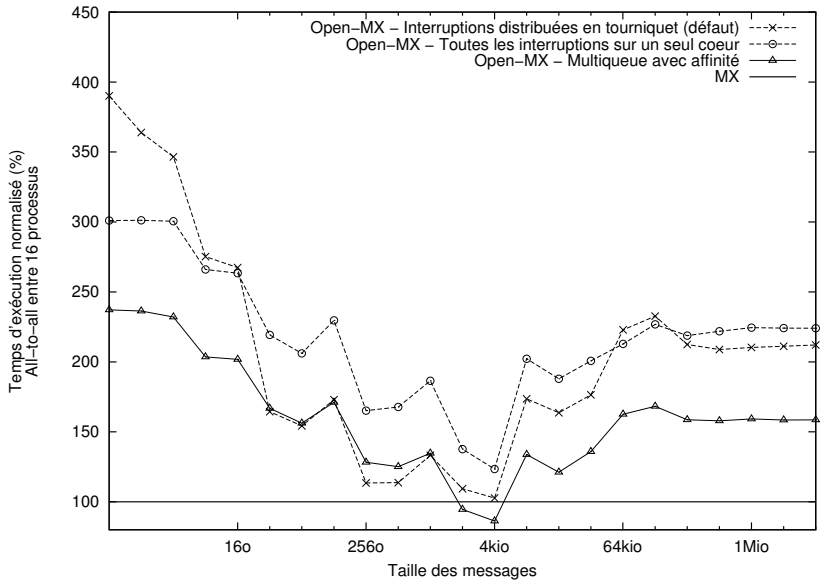


FIGURE 2.4 – Performances d'une opération collective All-to-all entre 16 processus selon la répartition des interruptions.

ne disposant pas de cartes réseau capables de distribuer les paquets reçus sur différents cœurs, mais elle illustre bien la nécessité de tenir compte des affinités dans les couches réseau modernes.

2.3.3 Adaptation des délais d'interruption

Un des autres axes importants dans la réception des paquets réseau est la notification de leur arrivée. Les réseaux rapides du HPC ont la capacité de déposer directement les paquets reçus dans la mémoire de l'application, ce qui permet à cette dernière de tester explicitement leur arrivée. Sur un réseau traditionnel, cette fonctionnalité est impossible car la carte réseau ne sait pas où se trouve la mémoire de l'application. Le mécanisme de notification se base donc essentiellement sur des interruptions de la carte lorsqu'elle souhaite signaler de nouveaux paquets au système d'exploitation.

Cette stratégie impose un compromis : si on notifie immédiatement l'arrivée d'un paquet, le traitement des interruptions peut surcharger le processeur central. À l'opposé, si la carte réseau attend pour signaler plusieurs paquets d'un coup (*Interrupt Coalescing*), la latence observée au niveau de l'application augmente car des paquets sont arrivés sans avoir été signalés. Dans le cas des flux réseau classiques tels que TCP, les faibles latences, de l'ordre de la microseconde, sont rarement nécessaires. Le coalescing est donc couramment utilisé par les cartes Ethernet, et l'interruption n'est déclenchée que quand un certain délai (souvent quelques dizaines de microsecondes) est atteint, ou lorsqu'un certain nombre de paquets a été reçu.

Même si ces limites sont configurables logiciellement, elles sont difficilement applicables au HPC car les petits messages nécessitant une très faible latence ne veulent aucun coalescing, tandis que les gros messages nécessitant une grande bande passante nécessitent du coalescing pour réduire la charge due aux interruptions. Derrière cette incompatibilité se cache en fait la différence entre les protocoles basés sur des flux, comme TCP, pour lesquels l'*Interrupt Coalescing* a été conçu, et les protocoles de passage de messages comme MPI.

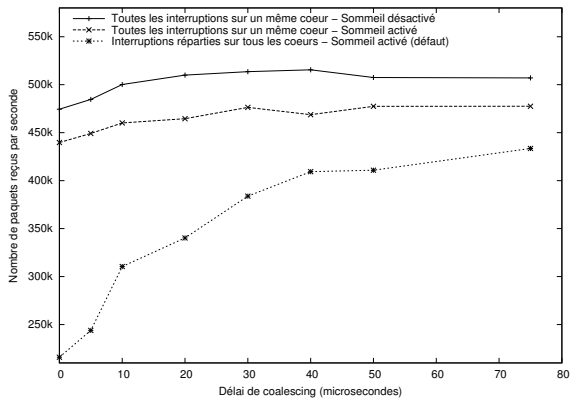
Nous avons donc proposé avec Nathalie Furmento un coalescing adapté à MPI, en utilisant la structure des communications sous forme de messages. L'idée est qu'un petit message est généralement indépendant et l'application bénéficiera d'une notification rapide, tandis qu'un paquet faisant partie d'une gros message n'impose une notification que quand tous les paquets correspondants sont arrivés. Nous avons donc modifié le micro-programme de la carte réseau pour tenir compte de ce structuration en messages au moment de décider de coalescer les interruptions ou non. Là aussi, ce travail n'impose que l'ajout d'une dizaine de lignes de code.

Ces travaux sont détaillés dans [CI8] et leur apport est rapidement présenté par les figure 2.5. Ils permettent de bénéficier d'un mode d'interruption adapté pour les cas sensibles à la latence ou au débit.

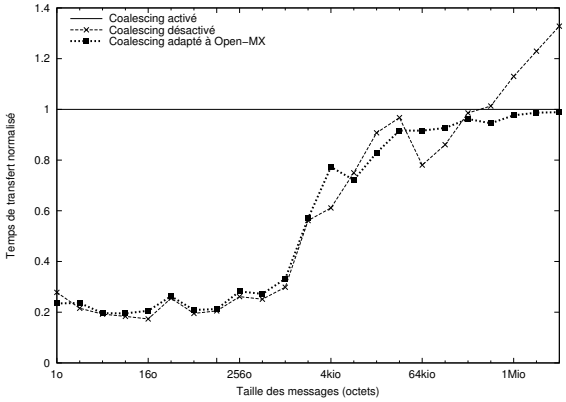
Les fabricants de carte réseau ont récemment introduit une extension nommée *Adaptive Coalescing* qui fait varier les seuils selon la charge du réseau. Cette technique permet de réagir aux variations d'intensité des flux quand la granularité est au moins de l'ordre de la dizaine de millisecondes. Cette stratégie semble bien fonctionner pour les applications parallèles alternant clairement des phases avec peu de petits messages et d'autres avec beaucoup de gros messages. Par contre, quand le schéma de communication est plus complexe, notre solution reste plus adaptée grâce à sa très fine granularité.

2.4 Bilan et perspectives

La convergence annoncée entre les réseaux rapides du HPC et les réseaux traditionnels comme Ethernet n'a pas vraiment eu lieu. Il subsiste un fossé très large entre ces deux mondes



(a) Nombre de messages de 128 octets pouvant être traités par le récepteur selon le délai de coalescing des interruptions.



(b) Débit relative d'un ping-pong selon le type de coalescing.

FIGURE 2.5 – Intérêt de l'*Interrupt Coalescing* adapté à la structure des messages Open-MX.

comme je l'ai montré dans la section 2.1.3.

InfiniBand reste très largement cantonné au HPC et a peu évolué depuis 5 ans, si ce n'est pour améliorer ses performances brutes et compenser certaines lacunes de son modèle. Les supercalculateurs continuent à utiliser des réseaux spécialisés, tandis que les autres domaines utilisent Ethernet. Ethernet a de plus reçu des améliorations significatives aux niveaux matériel et logiciel (comme CEE, TSO, LRO, etc.), et a pénétré le marché du stockage avec FCoE. C'est en partant de cette constatation que nous avons mené nos travaux sur Open-MX qui ont confirmé que MPI pouvait être implémenté plus efficacement sur Ethernet en reprenant des idées des réseaux rapides.

2.4.1 Des travaux fructueux

La collaboration contractuelle avec Myricom initiée en 2007 a permis d'explorer l'adaptation des réseaux Ethernet aux protocoles de passage de messages. J'ai montré que ce modèle permettait des performances satisfaisantes, inférieures à celles des réseaux rapides spécifiques et onéreux, mais supérieures à celles des couches TCP/IP habituelles. Mais j'ai aussi mis en évidence les lacunes du modèle, en particulier du coté récepteur, et proposé des solutions en adaptant différentes technologies modernes.

Les mécanismes de déport de copies mémoire ont permis de masquer les copies additionnelles imposées. Ensuite nous avons montré qu'il était facile d'adapter le filtrage des paquets entrants en tenant compte de la structure en messages et de leur destinataire. Cela autorise, d'une part, l'amélioration de l'utilisation du cache en regroupant les traitements et, d'autre part, l'optimisation de la notification des réceptions sans sacrifier soit la latence soit le débit.

Ces travaux ont donné lieu à de nombreuses publications dans des conférences et journaux internationaux [RR2, WI3, CI5, WI2, CI8, CI7, RI3, RI2].

La fin de ces travaux de recherche a conduit au fort ralentissement du développement d'Open-MX, le rendant moins populaire depuis quelques années. Mais il reste encore utilisé par quelques entreprises et universités internationales pour améliorer les performances de MPI sur Ethernet dans des grappes d'une dizaine de nœuds. [18]

18. Basement Supercomputing utilise encore Open-MX en 2014 dans sa plate-forme Limulus (`http://limulus.basement-supercomputing.com`) qui vise à spécifier un cluster de calcul personnel

2.4.2 Des mécanismes plus portables et accessibles ?

Mon objectif initial de rendre les mécanismes de communications locales des réseaux spécialisées plus portables et accessibles s'est décliné en deux points.

D'une part, j'ai effectivement mis en place l'interface de programmation de *Myrinet Express* au dessus de n'importe quel matériel Ethernet standard. L'idée fonctionne en utilisant des mécanismes de l'implémentation officielle MX, et peut par ailleurs être améliorée si certaines fonctionnalités additionnelles sont disponibles dans le matériel.

D'autre part, j'ai exposé ces travaux dans une interface de programmation réseau répandue. Si l'interface MX n'est pas parfaite (on y reviendra plus bas), elle a le mérite d'être déjà supportée par toutes les implémentations MPI majeures, qui sont stables, efficaces et proposent toutes les fonctionnalités de MPI. [19] C'est un avantage certain pour Open-MX là où des projets concurrents proposent une n-ième nouvelle interface de programmation réseau et doivent développer une nouvelle implémentation MPI associée.

2.4.3 De la nécessité d'avoir des interfaces génériques, portables et simples

La principale lacune que j'ai rencontrée lors du développement d'Open-MX est son interface de programmation trop centrée autour du matériel de Myricom. MX a été conçu pour être proche de MPI mais son modèle à base de connexions créées dynamiquement et de files d'événements centralisées convient également bien à d'autres applications, en particulier le stockage de fichiers (avec PVFS ou Lustre notamment [CI10, RR1]). Cette interface a d'ailleurs été copiée par Mellanox et QLogic pour offrir des fonctionnalités similaires dans MXM et PSM. Cependant le modèle de MX est particulièrement proche des fonctionnalités offertes par le matériel sous-jacent, en particulier le déport direct des notifications d'événements en espace utilisateur. Ces contraintes ont été fortes pour la conception d'Open-MX [20] et m'a conduit à réfléchir à une interface de programmation plus générique, que ce soit au niveau du support matériel en dessous ou au niveau des besoins des applications au dessus.

L'apparition des réseaux rapides pour le HPC s'est accompagnée de nombreuses nouvelles interfaces de programmation très spécifiques (Elan, SiSCI, GM...). InfiniBand a apporté une interface plus générique, les Verbs, espérant ainsi être utilisé par de nombreuses potentielles applications, même en dehors du HPC. Cependant, les Verbs sont étroitement liés au modèle de communication RDMA, et l'interface de programmation est très compliquée. Elle contient notamment un très grand nombre de concepts et objets (*Context*, *Queue-Pair*, *Protection Domain*, etc.) qui imposent un très gros travail aux développeurs. À titre d'exemple, la table 2.2 montre que les Verbs nécessitent entre 3 et 10 fois plus de code que les autres réseaux dans l'implémentation Open MPI. Même si ces chiffres ne sont pas forcément très représentatifs, ils laissent tout de même présager un travail de maintenance ou débogage beaucoup plus important sur InfiniBand.

Même si les implémentations MPI ont été portées sur InfiniBand et des améliorations ont été apportées pour ne plus être trop contraint par le modèle RDMA [21], l'adoption de l'interface Verbs dans d'autres domaines que le HPC reste cependant très limitée. Le succès d'une interface de programmation est lié à sa simplicité car la masse critique ne peut pas être atteinte en

basé sur des technologies ouvertes.

19. La révision 3.0 du standard MPI propose plus de 300 fonctions !

20. La collaboration contractuelle avec Myricom imposait une compatibilité entre MX et Open-MX aux niveaux bibliothèque et protocole afin de pouvoir les faire interopérer dans le système de stockage des machines BlueGene/P [RI2].

21. La *Shared Receive Queue* a par exemple permis de ne plus imposer une file de réception par processus distant, modèle qui ne pouvait pas passer à l'échelle des grandes applications parallèles.

BTL	Lignes de code
Elan	1 630
MX/ Open-MX	2 326
Portals	2 462
Sockets (TCP)	4 428
Sockets (SCTP)	5 160
UDAPL	6 166
OpenIB (Verbs)	15 521

TABLE 2.2 – Nombre de lignes de code dans le pilote (BTL) pour les différentes piles réseau supportées par Open MPI 1.6.

se limitant à quelques experts de la programmation réseau. Le succès des sockets est notamment dû à la simplicité de l'interface. Quant à MPI, même s'il définit plus de 300 fonctions, la plupart des utilisateurs n'utilise qu'une vingtaine d'entre elles.

L'OpenFabrics Alliance qui promeut les technologies InfiniBand s'est d'ailleurs rendue compte du problème. Ils savent que non seulement l'interface Verbs est peu appréciée des utilisateurs, mais pas non plus des constructeurs. En effet, c'est une interface (trop) bas niveau, qui collait bien aux premiers matériels InfiniBand mais a dû être pervertie pour s'adapter aux fonctionnalités matérielles récentes (XRC [22], Cisco `usnic` [23], etc.). Le fait que Mellanox pousse de plus en plus l'interface alternative MXM [24] montre également la prise de conscience du problème des Verbs. Des réflexions sont également menées sur l'interaction entre le système d'exploitation et l'espace utilisateur pour peut-être enfin résoudre proprement les problèmes d'enregistrement mémoire, comme évoqué dans la section 2.3.1.

Cependant, standardiser une version améliorée des Verbs va être un processus long et difficile. Il faudra convenir aux implémentations matérielles très variables, tout en satisfaisant les besoins des utilisateurs, et sans imposer trop de modifications aux applications existantes.

2.4.4 Quelle interface de programmation pour les réseaux du futur ?

Il y a une dizaine d'années, la multiplicité des technologies matérielles pour les réseaux du HPC avait engendré une jungle des interfaces de programmation réseau (MX et GM pour Myrinet, Verbs pour InfiniBand, Elan et Tports pour Quadrics, PSM pour Qlogic, etc.). Plus récemment, on pouvait espérer que l'hégémonie d'InfiniBand allait simplifier les choses en offrant enfin une interface centralisée. La tendance des constructeurs à déporter certaines fonctionnalités dans des boîtes noires matérielles a par ailleurs pu donner l'impression qu'il restait peu à faire en recherche dans le domaine des réseaux pour le HPC. Malheureusement, j'ai expliqué dans la section précédente que les Verbs ne convenaient plus aux évolutions récentes des technologies proches d'InfiniBand. Et les supercalculateurs continuent à utiliser les interfaces spécifiques (PAMI pour BlueGene/Q, GNI pour Cray, etc.).

En parallèle de ces problèmes bas-niveau, les interfaces de programmation haut niveau ne convergent guère. Le standard MPI devient de plus en plus compliqué (plus de 300 fonctions dans MPI 3.0) et de nombreuses propositions pour les modèles PGAS sortent régulièrement. De plus, avec la convergence annoncée des réseaux du HPC avec le stockage, les datacenters, voire le *High-Frequency Trading*, les besoins s'élargissent.

22. L'*eXtended Reliable Connection* est un *hack* des cartes Mellanox pour supporter beaucoup plus de connexions sans gaspiller trop de mémoire.

23. L'interface Verbs a dû être modifiée pour autoriser des tailles de paquets quelconques comme sur Ethernet au lieu des MTUs spécifiques à InfiniBand (seulement 5 puissances de 2).

24. Mellanox ne supporte plus l'implémentation de Open MPI sur Verbs mais uniquement celle sur MXM.

Le problème de la définition d'une interface réseau globale pour la haute performance n'est donc malheureusement toujours pas résolu comme certains pouvaient l'espérer. La multiplicité des interfaces pour réseaux rapides, non portables et pas assez génériques et/ou simples, a conduit beaucoup de développeurs à implémenter une couche d'abstraction pour pouvoir utiliser tous les réseaux de manière uniforme. Open MPI propose ainsi l'interface interne BTL, PVFS utilise BMI, Lustre a son LNET, etc. En observant ces interfaces, on remarque qu'elles sont souvent assez similaires. Un travail a donc été mené par notamment *Oak Ridge National Laboratory*, Myricom et Cisco pour définir une nouvelle interface de communication satisfaisant tous ces besoins, tout en restant simple et portable, CCI (*Common Communication Interface* [6] [25]).

L'interface reprend l'idée des connexions dynamiques et des files d'événements centralisées (appréciées dans MX). Les gros messages peuvent être transférés par RDMA comme dans les Verbs d'InfiniBand. Par contre les petits messages utilisent un modèle complètement différent, inspiré du comportement interne de la plupart des piles de passage de messages : la bibliothèque reçoit les petits messages dans ses zones mémoire propres puis les *prête* à l'application, qui pourra les lire directement, ou les copier ailleurs si nécessaire. Cette interface est en cours de développement et des applications de différents domaines, HPC, finance (*High-Frequency Trading*), sont portées dessus. Nous faisons désormais partie de l'initiative afin d'y apporter nos compétences pour l'implémentation au dessus d'Ethernet et notre connaissance des besoins des applications de stockage [Th1]. Certaines idées de CCI sont évoquées dans les réflexions autour de l'amélioration de Verbs évoquées dans la partie précédente. Mais il existe encore d'autres initiatives similaires comme par exemple LLC au Japon. Reste à savoir si l'une d'elle parviendra à convaincre la communauté. La résultat ne pourra pas être parfait en terme de performance, mais est-ce vraiment si grave de perdre 5% de débit ou de latence si on peut enfin espérer avoir une interface réseau générique ?

25. http://www.cci-forum.org.

Chapitre 3
Communication intra-nœud haute performance et portable

Les communications intra-nœud (c'est-à-dire entre processus s'exécutant sur un même nœud ou par extension dans une même instance de système d'exploitation) sont un des enjeux des bibliothèques de communication pour calcul haute performance depuis longtemps. Les serveurs biprocesseur sont très largement utilisés dans les grappes de calcul depuis la fin des années 90, et ils sont toujours une solution efficace pour les supercalculateurs modernes. Le centre de calcul de Leibniz a par exemple inauguré en 2012 la grappe SuperMUC qui assemble 9216 serveurs bi-processeur pour dépasser 3 petaflop/s. [1]

Avec la généralisation des processeurs multicœurs depuis mon post-doctorat, l'importance des communications intra-nœud s'est accrue. Et elle est mise en exergue par le comportement des applications et des utilisateurs, qui font en sorte que les processus communiquent plus avec leurs voisins qu'avec les processus distants, pour des raisons évidentes de performance. Le placement des applications est en effet de plus en plus important, on y reviendra dans le chapitre 4. La part de communications locales est donc beaucoup plus importante qu'il n'y paraît, et leur performance devient critique.

Ce travail entre dans mon premier grand axe de recherche présenté en section 1.3.1 en tentant de rendre plus largement disponibles certaines optimisations des communications intra-nœud.

3.1 Des solutions trop dépendantes du matériel

L'optimisation des communications locales est un sujet de recherche depuis plus de trente ans. Les adresses IP commençant par 127 leur sont réservées, et de nombreuses applications utilisent des sockets pour les échanges entre processus locaux. L'interface loopback est désormais un court-circuit logiciel de la pile IP, et les grappes l'ont utilisé initialement pour les communications MPI intra-nœud. Si certaines améliorations portables ont été proposées, la plupart des innovations dans le domaine sont venues des réseaux spécialisés qui ont une fois de plus fait étalage de leur non-portabilité et non-standardisation. Comme nous l'avons détaillé dans le chapitre 2 à propos des communications inter-nœud, nous allons présenter ici notre travail sur la portabilité et l'indépendance du matériel pour les stratégies optimisées de communication intra-nœud.

1. SuperMUC était 10ème au classement Top500 en novembre 2013, http://www.lrz.de/services/compute/supermuc/systemdescription/.

3.1.1 Des optimisations spécifiques au matériel

J'ai expliqué en section 2.1 que les réseaux rapides sont très présents dans le calcul haute performance depuis 15 ans et ont été un vecteur important d'innovation. Leurs cartes réseau très puissantes peuvent lire/écrire en mémoire des processus directement par DMA. Un moyen efficace de transférer des données entre processus locaux consiste donc à utiliser ces cartes pour ce faire. Cette stratégie (nommée *Loopback matérielle* dans la figure 3.1) est encore intéressante de nos jours en terme de performance [43]. Mais elle implique 2 transferts par DMA à travers les bus mémoire et d'entrées-sorties, ce qui cause des congestions impactant les performances des communications inter-nœuds et de l'application en général.

Un modèle purement logiciel semble plus intéressant car il court-circuite complètement la carte réseau et le bus d'entrées-sorties. L'idée est alors de bénéficier des pilotes dédiées aux réseaux haute performance. Comme l'administrateur doit charger ces pilotes, les constructeurs en ont profité pour y ajouter une implémentation modifiée des communications locales par loopback logicielle [27]. Cette idée est nommée *Assistance du pilote dépendant du matériel* sur la figure 3.1. Il s'agit d'utiliser le système d'exploitation pour faire une copie directe entre processus. Cependant, ces implémentations sont complètement dépendantes du matériel réseau, et donc non portables vers d'autres plates-formes, alors que les communications locales en sont a priori indépendantes. Par ailleurs, elles ont été développées en ciblant le cas habituellement testé, le *Ping-Pong*, afin de mieux convaincre les clients, mais sans s'intéresser aux autres types de communication (collectives, accès mémoire à distance...).

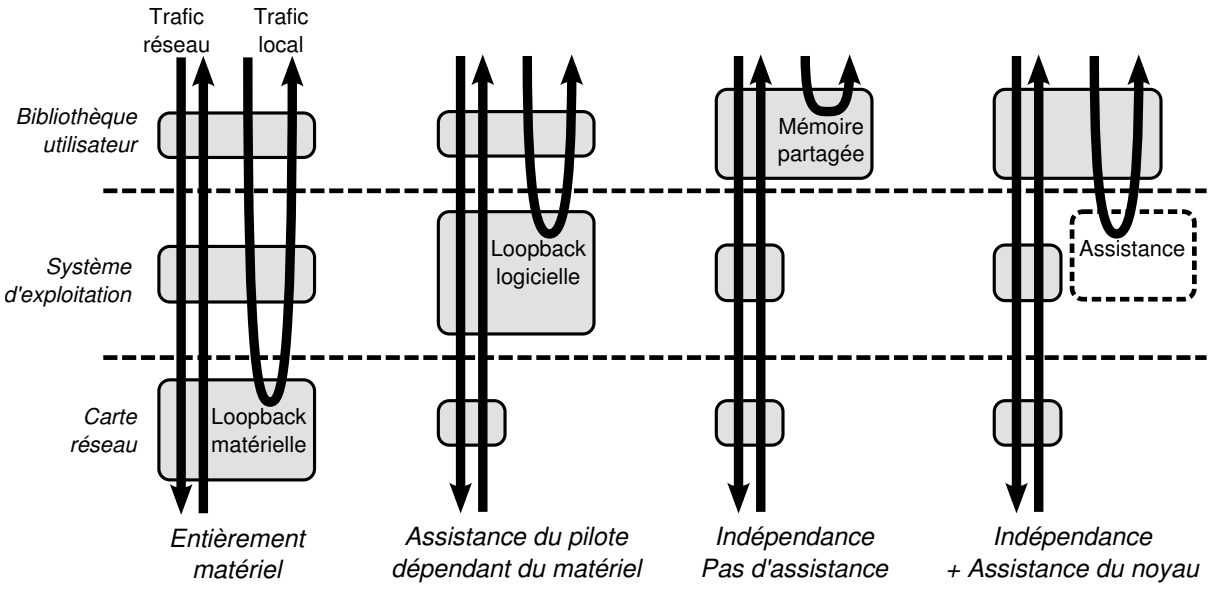


FIGURE 3.1 – Comparaison des possibles implémentations de communication intra-nœud, avec dépendance au matériel réseau ou non, avec assistance du noyau ou non.

Des extensions du système d'exploitation ont également été proposées mais elles étaient là aussi spécifiques à certaines plates-formes. Par exemple SMARTMAP permet l'accès direct à la mémoire des autres processus grâce aux mécanismes du micro-noyau Catamount des machines Cray XT [13]. Le module XPMEM pour le noyau Linux permet de projeter la mémoire d'autres processus dans notre mémoire virtuelle mais l'idée est là aussi limitée aux machines SGI. Bien qu'elles ne dépendent pas de technologies réseau, ces idées ne sont pas applicables aux grappes de calcul standards.

Toutes les implémentations MPI doivent fournir un support pour les communications lo-

cales, soit matériellement, soit logiciellement. Si un réseau rapide est disponible, son support spécialisé pourra être utilisé, sinon l'implémentation MPI devra s'en occuper. Cela mène à des situations inattendues comme la nécessité d'ajouter cette fonctionnalité dans notre bibliothèque Open-MX car MX la fournit et l'implémentation MPICH-MX suppose qu'elle est disponible. Cette fonctionnalité dans Open-MX est d'ailleurs à la base des travaux sur KNEM que je vais présenter ci après [CI4].

3.1.2 Besoin d'une solution portable

La figure 3.1 résume les solutions existantes pour les communications intra-nœud. La dépendance matérielle est pratique car elle bénéficie de la carte réseau et de son pilote dédié pour implémenter une loopback optimisée. Cependant ces solutions ne sont pas portables à d'autres plates-formes. Les implémentations MPI ont toujours à fournir une implémentation au cas où aucun réseau spécifique n'est présent.

Les systèmes d'exploitation offrent différentes interfaces de communication inter-processus telles que les sockets (à travers l'interface loopback) ou les tubes sous Unix. Mais elles ne sont pas optimisées pour le passage de messages et impliquent des appels-système qui accroissent significativement la latence. [2] Les implémentations MPI utilisent désormais un autre type de communication inter-processus : la mémoire partagée, représentée dans le cas *Pas d'assistance* sur la figure 3.1. Cette solution portable consiste à avoir l'émetteur qui écrit le message dans un tampon de mémoire partagée avant que le récepteur ne le copie à son tour dans sa zone de réception. L'utilisation d'opérations atomiques et de stratégies pipelinées permet des optimisations menant à des latences très faibles [17].

Cependant, utiliser cette stratégie pour des grands messages (centaines de kilooctets ou plus) provoque de la pollution de cache, du gaspillage de bande passante mémoire, et de la consommation de temps processeur à cause des copies mémoires nécessaires [16]. La performance pour ces gros messages peut bénéficier significativement d'un modèle basé sur une seule copie directe entre processus. Cette idée a été démontrée par des solutions dépendantes du matériel au détriment de la portabilité, comme expliqué plus haut.

J'ai cherché à proposer une solution générique qui résout ces problèmes sans dépendre du matériel, en suivant l'idée de l'*assistance du noyau* sur la figure 3.1. Je me suis focalisé sur les grappes Linux qui représentent plus de 90% du Top500, et probablement une part encore plus large des grappes de la planète. Ma solution nommée KNEM [3] a été développée en collaboration à l'Argonne National Lab où l'implémentation MPICH est développée. Ce travail a également été intégré dans l'implémentation Open MPI par la suite et a donné lieu à une collaboration autour des opérations collectives intra-nœud que nous détaillerons plus loin. KNEM est désormais supporté par MPICH depuis la version 1.1.1 et Open MPI depuis 1.5. Toutes les implémentations dérivées telles que MVAPICH ou Bullx MPI en bénéficient donc également. Je vais présenter dans les sections suivantes comment nous avons conçu KNEM pour offrir plus de flexibilité que ses concurrents grâce au support des copies asynchrones, des tampons mémoire non contigus, et du déport des copies sur moteur DMA.

3.2 Au-delà des opérations bilatérales point-à-point

Les solutions existantes de copies directes entre processus ont montré l'intérêt de l'idée mais sont restées fortement liées à certaines technologies. Je présente maintenant la façon dont

2. Un appel-système coûte environ 100 ns sur les serveurs modernes alors qu'une communication intra-nœud optimisée peut atteindre une latence de 200 ns.
3. KNEM est disponible sous licence BSD à `http://runtime.bordeaux.inria.fr/knem`.

j'ai rendu cette idée portable tout en l'étendant pour bénéficier à beaucoup plus que les habituelles communications point-à-point (*Send-Receive*) entre 2 processus, en particulier les opérations collectives.

Cette flexibilité vise aussi à casser l'habitude récurrente et malsaine de ne cibler que le micro-benchmark habituel : le ping-pong MPI. Les solutions non portables des constructeurs sont en effet souvent réservées aux communications point-à-point, car cela suffit pour le marketing. Malheureusement, les vraies applications parallèles ont par exemple également besoin de telles innovations pour les opérations collectives, mais c'est moins facile à mettre en avant pour les constructeurs !

3.2.1 Design général

La conception d'une interface noyau assistant les implémentations MPI pour les communications locales soulèvent plusieurs questions. Tout d'abord il faut garder à l'esprit que les fonctionnalités ajoutées au système d'exploitation impliquent du code s'exécutant en mode privilégié. Il faut donc veiller à ne déporter dans le noyau que le code qui a besoin d'y être afin de limiter les risques de failles de sécurité. Par ailleurs, la séparation logique entre l'espace noyau et les applications empêche le premier de tout savoir sur ce que font et souhaitent les secondes. Il est donc important de bien placer cette séparation logique.

KNEM est un module noyau pour des communications MPI. On peut donc envisager qu'il offre une interface proche de celle de MPI, en particulier le *Matching*. [4] C'est une solution qui a été initialement choisie par le logiciel LiMIC [41] mais finalement abandonnée [40]. En effet, elle implique de dupliquer dans le noyau une partie significative des implémentations MPI : les files de messages entrants et de requêtes de réception, l'algorithme de matching, la gestion des différents modes de communications (synchrone ou non, prêt ou non, bloquant ou non...). Il vaut mieux garder le *Matching* existant en espace utilisateur, ce qui réduit la quantité de code privilégié. KNEM expose donc une interface de programmation simple qui suppose que l'espace utilisateur a déjà réalisé le matching.

La question suivante vise l'identification des régions mémoires impliquées dans le transfert. En effet, une copie directe entre processus imposent de savoir où se trouve les données du processus distant et comment elles sont organisées en mémoire. Certaines implémentations comme LiMIC et SMARTMAP impose au processus effectuant la copie de connaître ces informations. Il doit donc les récupérer préalablement avant de pouvoir lancer la copie, en particulier si la région mémoire est découpée en beaucoup de petits morceaux. [5] De plus, un processus malicieux pourrait modifier ces informations avant d'accéder à la mémoire. Notre approche dans KNEM élimine tous ces problèmes en masquant ces informations à l'intérieur d'un *Cookie* déclaré au système d'exploitation par le processus cible. Le processus accédant aux données ne pourra accéder qu'à celles décrites dans ce cookie. Par ailleurs, ce modèle permet des optimisations en cas de réutilisation multiples d'une même région mémoire.

3.2.2 Interface Send-Receive

L'interface initiale de KNEM peut donc se résumer de la manière suivante :

— Le processus émetteur déclare une zone d'émission (contigüe ou non) au pilote KNEM et reçoit un identifiant Cookie en retour.

4. Le matching consiste en l'association de chaque message entrant avec une requête de réception qui lui correspond.

5. Les datatypes MPI permettent de décrire des régions mémoires fortement non contigües, par exemple une colonne de matrice.

— Le processus émetteur passe ce Cookie aux autres processus auxquels il veut envoyer des données.
— Le processus récepteur passe le Cookie reçu ainsi qu'un descripteur de sa zone de réception au pilote KNEM qui se charge de faire la copie selon l'organisation mémoire décrite dans le cookie et celle de la zone de réception.

Ce modèle a été facilement intégré dans les implémentations MPICH et Open MPI car le Cookie peut être échangé dans le message de contrôle utilisé pour le rendez-vous entre les deux processus. Quand le cookie est reçu, l'implémentation MPI effectue le matching puis en déduit la requête de copie à passer au pilote KNEM. Ce modèle est décrit dans la figure 3.2 et détaillé dans [RI4].

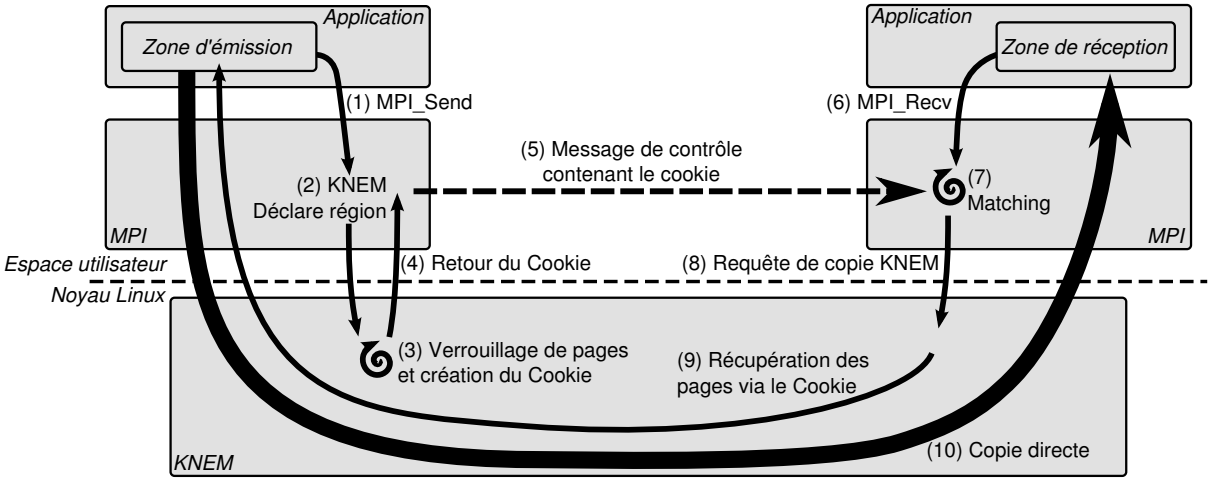


FIGURE 3.2 – Implémentation d'un Send-Recv MPI au dessus de KNEM.

Ce modèle a initialement été implémenté dans MPICH dans le cadre d'une collaboration avec l'équipe Radix du Argonne National Lab. Le nom KNEM vient d'ailleurs de *Kernel Nemesis*, Nemesis étant le nom du cœur de l'implémentation MPICH 2. La figure 3.3 montre que Nemesis sépare la gestion des messages avant et après 64 ko. Avant, la mémoire partagée est utilisée, car c'est la technique permettant d'obtenir les meilleures latences (car aucun appel-système n'est impliqué). Après, différents LMT (*Large Message Transfer*) peuvent être utilisés pour transférer les gros messages. Un nouveau a été ajouté à MPICH pour utiliser KNEM [CI3].

3.2.3 Opportunités d'amélioration

Si l'interface de KNEM a permis de fournir à MPICH des communications optimisées pour les gros messages grâce à une copie directe, cette idée n'était pas révolutionnaire. La seule nouveauté consistait en l'indépendance de cette implémentation vis-à-vis du matériel sous-jacent. Nous avons poursuivi ces travaux en nous intéressant à d'autres types de communication que les point-à-point.

La figure 3.4 détaille l'exécution d'une opération collective Broadcast pour différentes implémentations. De nombreux algorithmes ont été proposés par le passé [66] et la plupart des implémentations MPI en choisit dynamiquement un adapté aux circonstances [61]. Le schéma montre que la disponibilité d'une copie directe peut effectivement supprimer une séquentialisation du processus racine et permettre une parallélisation complète des différents transferts. Les implémentations usuelles nécessitent des étapes supplémentaires car chaque transfert impose que l'émetteur ait fini d'écrire avant que le récepteur puisse lire et utiliser les données.

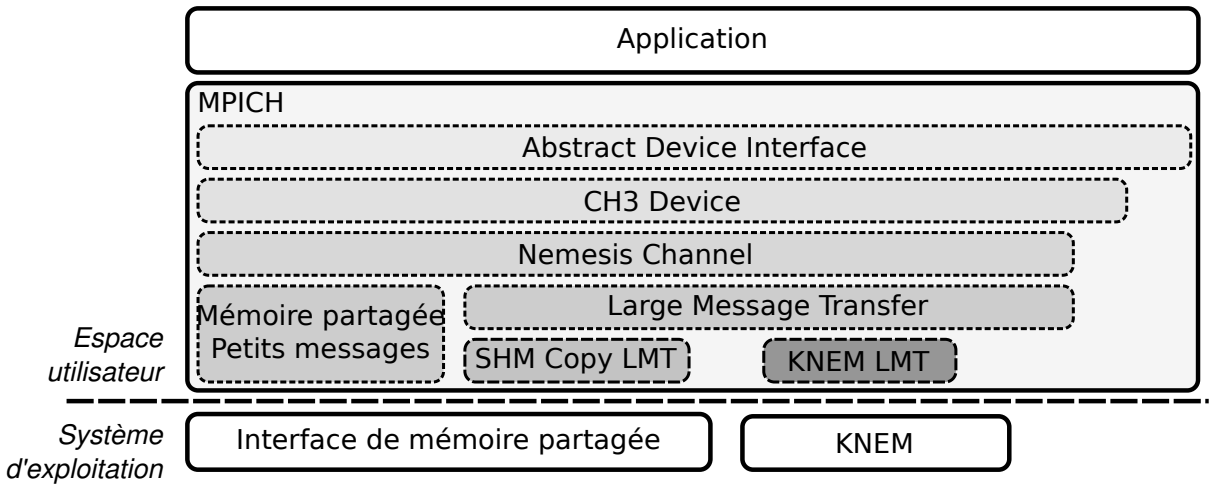
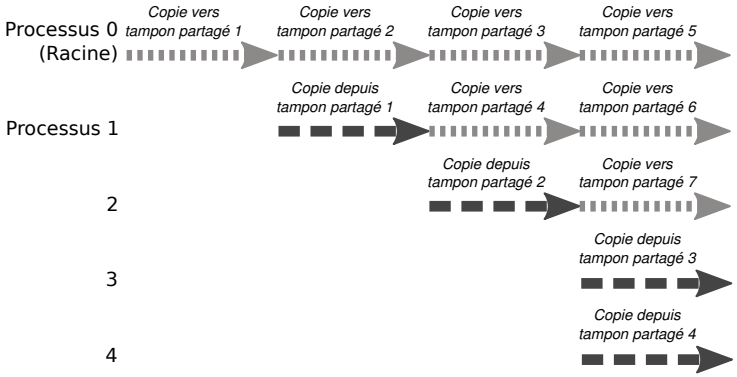
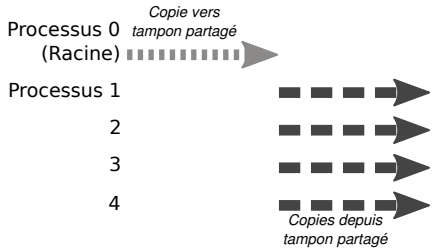


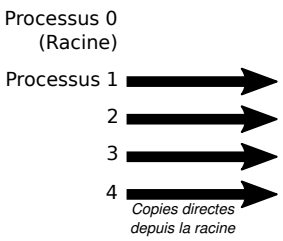
FIGURE 3.3 – Intégration de KNEM dans les communications point-à-point de MPICH.



(a) Arbre binaire basé sur des opérations point-à-point en mémoire partagée.

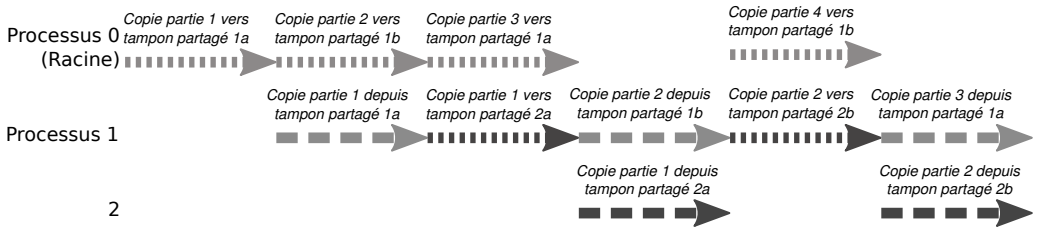


(b) Mémoire partagée entre tous les processus.

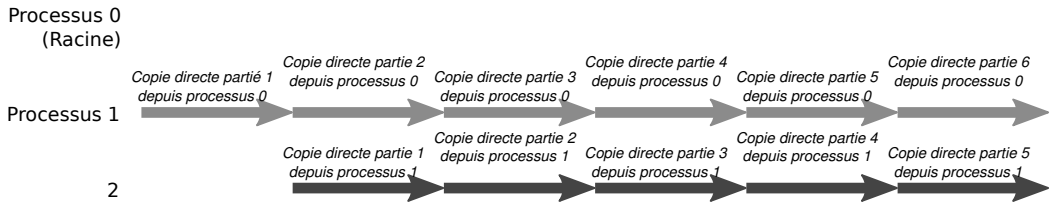


(c) Copies directes.

FIGURE 3.4 – Copies de données au cours de l'exécution d'un Broadcast selon l'implémentation.



(a) Avec des opérations point-à-point en mémoire partagée.



(b) Avec des copies directes.

FIGURE 3.5 – Copies de données au cours d'un transfert pipeliné (deux étages) par morceaux. Le processus 1 sert d'intermédiaire entre les processus 0 et 2.

La possibilité de ne transférer qu'une partie d'un message permet d'aller plus loin dans l'amélioration des opérations. Les algorithmes pipelinés à plusieurs niveaux sont également couramment utilisés dans les opérations collectives car ils réduisent le temps de démarrage. La figure 3.5 présente l'exécution d'un transfert entre deux processus avec un troisième jouant le rôle d'intermédiaire. Elle montre que la copie directe peut réduire significativement le temps de transfert global car le second transfert d'un morceau peut être recouvert par le premier transfert du suivant. Dans le cas usuel, le processus intermédiaire est impliqué dans les deux transferts et conduit donc les deux autres processus à patienter pendant la moitié du temps.

3.2.4 Accès à la mémoire distante

Les opportunités d'amélioration exposées dans la section précédente nous ont conduit à généraliser l'idée du *Cookie* initialement conçue pour les communications point-à-point *Send-Receive*. En effet, les opérations collectives impliquent de multiples transferts depuis ou vers la même région mémoire, ou une partie de cette zone. Le paradigme de programmation par accès mémoire à distance (RMA, *Remote Memory Access*) a des besoins similaires. Nous avons donc rendu possible l'utilisation du même Cookie plusieurs fois sans avoir à le redéclarer à chaque fois. Cela permet de factoriser les appels-système et le verrouillage mémoire nécessaires, et permet l'application de stratégies de type *Cache d'enregistrement* [65].

L'implémentation d'un Broadcast sur KNEM permet donc de ne déclarer la région d'émission qu'une seule fois pour tous les destinataires, choses que les concurrents comme LiMIC ne peuvent pas faire. Un Scatter bénéficie également du modèle puisque la même zone peut également être utilisée mais des parties différentes vont être lues par chaque destinataire. L'interface étendue de KNEM remplace donc le *Cookie* d'émission par **une région mémoire déclarée, qui peut être lue ou écrite, partiellement ou entièrement, par plusieurs processus plusieurs fois**.

La possibilité de lire ou écrire dans une région mémoire distante est très pratique car elle permet de contrôler quel(s) processus effectue la copie mémoire (chère), alors que c'était toujours le récepteur dans l'ancien modèle. C'est utile pour éviter la séquentialisation des copies indépendantes en retirant leur traitement du processus central limitant. Par exemple, une opé-

ration collective telle que Broadcast ou Gather sera lente si le processus racine séquentialise les copies, elles doivent plutôt être effectuées par les processus feuilles, en parallèle. Le Broadcast aura donc les récepteurs qui lisent depuis la racine, tandis qu'un Gather aura les émetteurs qui écrivent dans la racine.

Une autre justification pour l'interface KNEM étendue est la nécessité de **laisser les implémentations MPI optimiser leurs stratégies en utilisant toutes les informations de contexte disponiblee**. L'ancienne interface KNEM et ses concurrentes ne peuvent être utilisées que dans les couches point-à-point *Send-Receive* tout en bas des implémentations MPI. Même si les collectives ou RMA peuvent être implémentées par dessus, cette étape intermédiaire réduit les possibilités d'optimisation car la couche manipulant KNEM ne sait pas ce que font les couches supérieures. Faut-il déplacer la copie vers le processus émetteur ? La région sera-t-elle réutilisée ? L'interface KNEM étendue peut être utilisée directement par les couches supérieures sans avoir d'intermédiaire opaque. Ceci permet aux différents composants d'une implémentation MPI (point-à-point, collectives, RMA...) de tous utiliser KNEM directement et efficacement. Cette approche nous semble plus satisfaisante que le développement d'un module noyau dédié à chaque fonctionnalité MPI comme ce fût le cas pour les RMA dans MVAPICH [44].

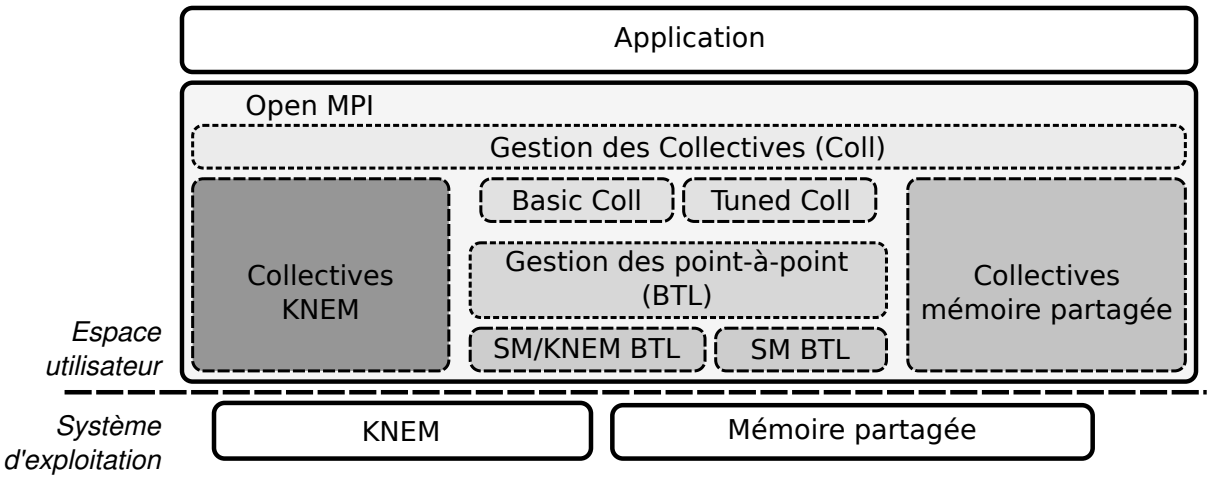


FIGURE 3.6 – Intégration de KNEM dans les communications point-à-point et collectives de Open MPI.

La figure 3.6 présente l'architecture des composants Open MPI et l'intégration de KNEM. Plusieurs composants fournissent des opérations collectives, par exemple *Basic* et *Tuned* qui les implémentent au dessus d'opérations point-à-point (BTL). Ces opérations point-à-point peuvent utiliser uniquement la mémoire partagée (BTL *SM*) ou la combiner avec KNEM (BTL *SM/KNEM*). D'autres composants implémentent les collectives directement sans passer par les point-à-point. Nous avons ajouté un tel composant utilisant directement l'interface KNEM étendue [CI12]. Ce composant optimise les opérations collectives en appliquant les idées présentées en section 3.2.3.

3.3 Performance et dépendance à la topologie matérielle

Je présente dans cette section une rapide évaluation des performances de KNEM, d'abord pour les communications point-à-point puis collectives. On pourra se référer aux publications [CI3, WI8, CI12, RI4] pour plus de détails.

3.3.1 Des performance point-à-point très variables

Les performances initiales de KNEM dans l'implémentation MPICH ont été détaillées dans l'article [CI3]. Nous avons mis en évidence un apport significatif de KNEM pour le transfert de gros messages MPI sur les architectures à bus mémoire centralisé où la réduction du nombre de copies mémoire est très importante. Depuis, les architectures NUMA se sont généralisées et la bande passante mémoire disponible a très largement augmenté. Si l'apport de KNEM sur les micro-benchmarks point-à-point s'est réduit, il est toujours très important quand le schéma de communication se charge, car la saturation de la mémoire impose de réduire les copies.

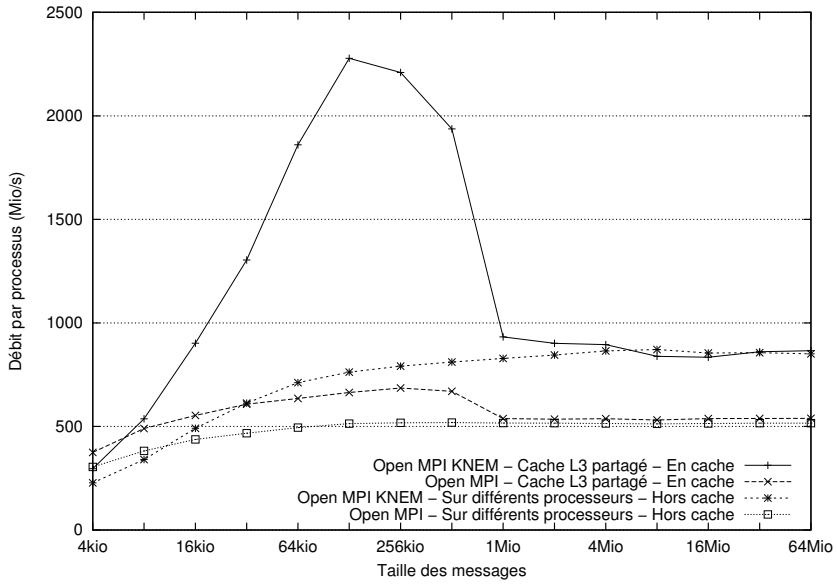


FIGURE 3.7 – Performance d'un *Ping-Ping* entre deux processus s'exécutant sur un machine à 4 processeurs AMD Opteron 8347HE quadricœurs.

La figure 3.7 présente la débit au cours d'un *Ping-Ping*, c'est à dire que deux processus s'échangent des messages simultanément. [6] Sur cette plate-forme, les gains observés lors d'un *Ping-Pong* utilisant KNEM sont limités car l'architecture parvient à gérer efficacement les deux copies du modèle à mémoire partagée habituel. Dans le cas du *Ping-Ping*, l'existence de deux messages simultanés faire ressortir la saturation du débit mémoire et montre donc un gain d'un facteur 2 pour les gros messages. De plus amples comparaisons sur architectures modernes sont présentées dans [RI4].

Un autre phénomène à noter sur la figure est l'influence du cache. Si un cache est partagé entre les deux processus communicants et/ou si le message est dans le cache du processeur émetteur, le coût de la double-copie est moindre. Les effets de cache doivent donc être pris en compte avant de décider si KNEM doit être utilisé. Nous avons travaillé sur cet aspect au cours de la thèse de Stéphanie Moreaud [53]. Définir des seuils de transition entre les différents stratégies de communication est un problème difficile. Il s'applique aux communications externes et internes, comme je l'ai expliqué dans l'article de vulgarisation scientifique [MS3]. En intra-nœud, il dépend des performances des différentes implémentations (double-copie à travers une mémoire partagée, copie directe avec KNEM, voire déport de la copie KNEM sur matériel spécialisé I/O AT), du placement des processus communicants, et du comportement

6. Un ping-ping consiste en un ping-pong d'un processus vers l'autre en même temps qu'un autre ping-pong en sens inverse.

de l'application (données émises récemment utilisées ou non, données reçues bientôt utilisées ou non).

Nous avons proposé dans [CI3] une heuristique de sélection de la stratégie dans MPICH et la même idée a été appliquée à Open MPI [49]. Cependant ces heuristiques dépendent de la charge de la machine et s'applique plutôt aux cas simples. Elles s'appliquent bien aux micro-benchmarks *Ping-Pong* qui sont toujours (trop) souvent utilisés pour comparer les implémentations. Par contre, quand le schéma de communication devient complexe et chargé, l'usage de KNEM doit être privilégié tôt pour réduire le gaspillage de bande passante mémoire.

3.3.2 Des collectives à adapter à la topologie matérielle

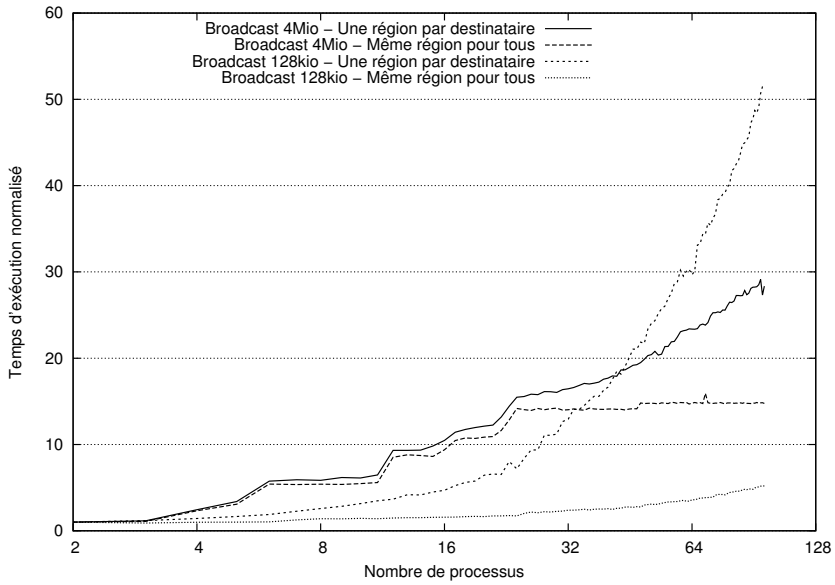


FIGURE 3.8 – Performance d'un Broadcast selon la réutilisation ou non d'une unique région mémoire pour tous les destinataires. Le temps est normalisé par rapport au coût d'un opération vers un seul destinataire. La plate-forme est constituée de 4 nœuds NUMA contenant chacun 4 processeurs hexacœurs (Intel *Dunnington X7460*).

La figure 3.8 présente l'intérêt de réutiliser une même région mémoire KNEM pour toutes les destinations d'une opération Broadcast. Quand le nombre de processus est faible, déclarer une seule région mémoire par destinataire ne pose pas de problème de performance. Pour un grand nombre de cœurs, on constate que la réutilisation d'une seule région décroît le coût d'un facteur 2 pour les gros messages de 4 Mo et 8 pour 128 ko. Ce résultat montre que le coût de création/destruction de multiples régions mémoire KNEM n'est pas négligeable (notamment à cause du verrouillage mémoire) et qu'il faut donc essayer de le factoriser. Des résultats allant dans le même sens ont été observés pour d'autres opérations collectives.

Pour terminer, nous présentons rapidement l'application à une application réelle. ASP est une implémentation parallèle de l'algorithme de Floyd-Warshall qui résout le problème du plus court chemin (*all-pairs-shortest-path*). Elle utilise de nombreuses opérations collectives, en particulier le Broadcast. Nous avons comparé son exécution avec l'implémentation habituelle des collectives dans Open MPI et un algorithme utilisant l'interface KNEM étendue, avec deux niveaux de pipeline. La vitesse d'exécution des Broadcast est améliorée de 48% grâce à nos travaux, et le temps global de l'application de 2%. Ces résultats sont détaillés dans [CI12] et d'autres applications ont depuis montré des gains notables [48].

3.4 Bilan et perspectives

Nos travaux autour de KNEM sont nés de la nécessité de rendre indépendant du matériel l'idée d'assister le passage de messages inter-processus par le noyau. Ils ont donné lieu à des collaborations avec les implémentations MPI les plus populaires et à plusieurs publications [CI4, CI3, WI8, CI12, RI4].

3.4.1 Des mécanismes plus portables et accessibles ?

Notre objectif initial de rendre les mécanismes de communications locales des réseaux spécialisées plus portables et accessibles s'est décliné en deux points.

D'une part, nous avons rendu ces idées indépendantes de la plate-forme matérielle. À l'opposé, les implémentations concurrentes sont liées au pilote d'une technologie réseau ou au système d'exploitation spécifique de certains supercalculateurs. Même si KNEM ne peut pas être actuellement considéré comme vraiment portable car il n'est disponible que sous Linux, cela couvre une très grande partie du calcul haute performance. [7] Par ailleurs, le portage de KNEM sous d'autres systèmes d'exploitation est tout à fait possible techniquement si le besoin apparaissait.

D'autre part, nous avons rendu ces idées accessibles à un panel beaucoup plus large d'opérations, et donc d'applications. En effet, toutes les implémentation concurrentes visent spécifiquement un type de communication, les point-à-point *Send-Receive*, notamment car c'est le benchmark que les constructeurs mettent en avant. Nous avons montré que fournir une interface plus flexible permettait d'améliorer significativement les autres opérations, en particulier les collectives.

3.4.2 Intégration de nos travaux dans les implémentations MPI

Le développement de KNEM est né d'une collaboration historique entre mon équipe de recherche et l'Argonne National Lab qui développe l'implémentation MPICH. Ce travail visait initialement à fournir dans MPICH une copie directe inter-processus indépendante du réseau matériel sous-jacent. La diffusion de ces travaux a rapidement conduit la communauté Open MPI à également intégrer le support KNEM. Comme la plupart des implémentations MPI sont dérivées de MPICH ou Open MPI, KNEM est désormais supporté dans une grande majorité de cas.

Le portage d'Open MPI a par ailleurs ouvert un nouvel axe de collaboration avec l'Université de Tennessee Knoxville (*Innovative Computing Lab*) autour de l'implémentation d'opérations collectives MPI adaptées aux machines modernes. Ce travail s'est traduit par une publication [CI12] et continue à être un axe de recherche [48, 47].

3.4.3 Enfin un embryon de support dans le noyau Linux

L'autre apport du succès de KNEM est que la communauté a pris conscience de l'intérêt d'utiliser le système d'exploitation pour assister les communications intra-nœud. Notre travail est arrivé au moment où les processeurs multicœurs se démocratisaient et rendaient donc cet axe de recherche très important.

L'étape suivante consiste à intégrer notre travail dans le noyau Linux afin que les administrateurs n'aient plus à installer notre module noyau privilégié. Nous avons veillé à sécuriser

7. 96,4% des supercalculateurs du Top500 utilisent Linux en Novembre 2013.

notre approche pour qu'elle soit utilisable en production. [8]

Les noyaux Linux récents (depuis 3.2) fournissent enfin une assistance aux communications intra-nœud sous la forme d'appels-système lisant ou écrivant la mémoire d'un autre processus. [9] Ce modèle est toujours trop restreint pour permettre tous les optimisations apportées par KNEM. Mais quand on connaît la difficulté à intégrer dans le noyau Linux des fonctionnalités pour le calcul haute performance, on peut se réjouir de ce pas en avant.

3.4.4 Copies ou tours de passe-passe avec la mémoire virtuelle

Une idée qui revient souvent quand on parle d'optimiser les copies mémoire consiste à utiliser la mémoire virtuelle pour déplacer virtuellement des données entre processus sans les copier physiquement. Si l'idée est très jolie sur le papier, elle est difficilement applicable ici. Tout d'abord, elle présente une grosse contrainte d'alignement vis-à-vis des pages de mémoire virtuelle : la zone source et la zone destination doivent être placées exactement de la même façon par rapports aux pages (en général 4 ko), ce qui n'est généralement pas le cas. Pour les communications réseau, on peut éviter le problème en intégrant cet alignement dans les transferts entre carte réseau et mémoire [56], mais c'est impossible pour les copies directes étudiées ici.

L'autre problème est le coût de ces manipulations de mémoire virtuelle. En effet, les changements de table des pages impliquent des modifications du TLB (*Translation Lookaside Buffer*), ce qui est très cher car cela doit être généralement notifié à tous les processeurs. Les développeurs d'Intel indiquent souvent qu'il est plus rapide de faire une copie mémoire car elle est selon eux *gratuite*. De mon point de vue, c'est une justification exagérée compte tenu de la pollution de cache, mais cela donne une bonne indication du coût des manipulations de table de pages.

Une autre solution parfois envisagée consiste à projeter de la mémoire d'un processus dans les autres afin de pouvoir directement y accéder [73]. Cependant, comme on ne peut projeter qu'une partie limitée, ce mécanisme doit être utilisé soit ponctuellement (ce qui impose les coûts de manipulation de table de pages à chaque communication) soit de manière persistante (pour factoriser ces coûts) ce qui impose de copier les données des autres zones dans la seule zone partagée. C'est pour ces raisons que j'ai préféré privilégier la piste des copies mémoire.

3.4.5 De l'importance de la localité

Tout au long de nos travaux autour de KNEM, nous avons été confronté au problème des effets de cache. L'existence des caches dans les processeurs, la possibilité qu'ils soient partagés entre les processus communicants ou non, et le comportement des applications vis-à-vis des données et de ces caches ont une grande influence sur les performances de communications intra-nœud. La démocratisation des architectures NUMA dans le calcul haute performance a également contribué à rendre les performances encore plus variables selon le placement des processus. Ces effets sont évidemment présents sur les opérations collectives complexes mais également sur les communications point-à-point simples.

Il est indispensable de nos jours de connaître la topologie des calculateurs pour espérer en exploiter les performances et comprendre le comportement des applications. Les outils tels que hwloc ont été conçus pour les attaquer [CI1]. Une modélisation des accès mémoire est nécessaire à la compréhension des performances des différentes stratégies de communication

8. Nous avons d'ailleurs montré que l'approche concurrente LiMIC créait une fiable de sécurité très facile à exploiter et permettait d'accéder à la mémoire de n'importe quel processus de la machine.

9. Appels-système `process_vm_readv` et `process_vm_writev`, parfois nommés *Cross Memory Attach* même s'il s'agit de copier et non pas d'attacher une zone mémoire distante.

présentée dans cette partie, et espérer choisir entre elles de manière adaptée. Mais cette idée s'applique aussi à d'autres problèmes que seules les communications intra-nœud. C'est ce que nous allons développer dans la seconde partie de ce manuscrit, en particulier dans le chapitre 4.

Chapitre 4
Comprendre la complexité des architectures modernes

La démocratisation des processeurs multicœurs et des architectures NUMA depuis une dizaine d'années rend l'exploitation de la puissance des supercalculateurs de plus en plus difficile. Alors qu'un code séquentiel était auparvant automatiquement accéléré à chaque nouvelle génération de micro-architecture grâce à une fréquence supérieure, c'est désormais le parallélisme qui règne.

Au-delà de la nécessité d'exhiber un fort degré de parallélisme dans les applications pour nourrir le matériel [1], il est devenu important de correctement répartir les tâches de calcul et leurs données pour maximiser la proximité des composants ayant des affinités. Pour ce faire, il faut d'abord savoir comment est organisé le matériel et comment il se comporte.

Cette modélisation des architectures est depuis longtemps un axe important de l'informatique car elle est indispensable pour concevoir des logiciels adaptés. Elle doit être suffisamment simple pour qu'on puisse l'utiliser dans des algorithmes de complexité raisonnable. Mais elle doit également être suffisamment détaillée pour fournir des informations pertinentes pouvant répondre aux critères de décision des algorithmes.

C'est l'objet de mes travaux autour du logiciel hwloc, qui est devenu en 3 ans la référence de la découverte de topologie matérielle et du placement de tâches, et de la thèse de Bertrand Putigny. Ces travaux se placent dans mon second grand axe de recherche présenté en section 1.3.2 et visant à faciliter l'exploitation des architectures de plus en plus complexes.

4.1 Portabilité des applications et topologie matérielle

La portabilité des performances est le *leitmotiv* de l'équipe-projet Runtime dans laquelle j'ai effectué mes recherches depuis mon arrivée à Inria en 2006. Le rêve d'applications parallèles s'exécutant immédiatement aussi efficacement sur une grappe de calcul bicœurs que sur une machine massivement multiprocesseur est devenu une utopie. D'une part parce que le modèle de programmation des architectures n'est pas toujours même, mémoire partagée ou distribuée, passage de messages ou accès mémoire à distance... D'autre part, parce qu'en supposant que le modèle idéal de programmation est le même, l'application devra être adaptée à l'organisation du matériel sous-jacent.

En effet, les applications sont de plus en plus souvent organisées de manière hiérarchique, avec par exemple un certain nombre de processus contenant un certain nombre de threads utilisant eux-mêmes des bibliothèques possiblement parallèles, par exemple les BLAS. Tous ces *certains nombres* doivent correspondre à la hiérarchie matérielle pour que l'application l'utilise efficacement. Cela impose donc de programmer les applications de manière flexible et de détecter à l'exécution l'organisation matérielle avant de s'y adapter.

1. Le supercalculateur Tianhe-2 en tête du Top500 de Novembre 2013 est composé de 384 000 cœurs généralistes (Intel Xeon) et plus de 2,7 millions de cœurs spécialisés (Intel Xeon Phi).

4.1.1 La course à la puissance... crête

La topologie des supercalculateurs était encore relativement simple il y a vingtaine d'années. Les supercalculateurs massivement parallèles pouvaient notamment être composés d'une architecture plate (SMP) où de nombreux processeurs accèdent directement à une mémoire centrale. L'émergence des grappes a imposé un modèle opposé, avec de nombreux nœuds très simples, le plus souvent biprocesseurs (et monocœurs à l'époque). Ces petites hiérarchies se satisfaisaient assez bien d'une programmation plate, par exemple un modèle MPI pur pour les grappes. La topologie n'était pas prise en compte dans la programmation de l'application. Elle n'était prise en compte que pour optimiser les communications MPI intra-nœud comme discuté dans le chapitre 3.

Avec l'apparition des machines NUMA, la hiérarchie mémoire a imposé de veiller au placement des tâches et des données. Si des choses ont été implémentées dans les systèmes d'exploitation pour tenter de garder les tâches et leurs données proches, les standards comme OpenMP fournissent toujours peu d'outils pour tenir compte de la topologie dans les applications. [2]

C'est surtout depuis une dizaine d'années, en particulier avec la généralisation des processeurs multicœurs puis des architectures NUMA dans les grappes, que la topologie des machines de calcul s'est considérablement complexifiée. En effet, on trouve aujourd'hui à l'intérieur des machines plusieurs niveaux de caches plus ou moins partagés, certaines unités de calcul partagées entre cœurs (Intel *Hyperthreading* ou unités flottantes AVX sur AMD), et une mémoire distribuée NUMA. J'ai présenté l'importance de toutes ces affinités dans l'article de vulgarisation scientifique [MS2].

Si la puissance théorique (crête) des machines continue à augmenter, elle est désormais liée à cette complexité. La difficulté pour les applications de s'y adapter empêche de combler le fossé entre puissance crête et puissance observée dans les applications. Et la tendance n'est pas à l'amélioration puisque les architectes matériels nous prévoient de plus en plus de cœurs pour les années à venir, voire la fin de la cohérence de cache matérielle.

Par ailleurs, l'augmentation du nombre de nœuds dans les grappes impose plusieurs niveaux de switchs pour les interconnecter efficacement. Il est désormais courant d'avoir dans les supercalculateurs modernes plus de 5 niveaux différents de hiérarchie. [3] Cette hiérarchie, en particulier le degré de chacun des niveaux, varie d'une plate-forme à une autre, imposant de réadapter le schéma applicatif. Que le modèle de programmation soit par thread, passage de messages, graphe de tâches... il est indispensable d'avoir des outils permettant de comprendre la topologie matérielle et de la décrire de manière portable afin que les applications puissent s'y adapter facilement.

4.1.2 Comment utiliser des informations de topologie ?

Le principal objectif de la connaissance de la topologie est le placement des tâches de calcul. On souhaite plonger dans la hiérarchie matérielle le schéma de l'application, qui peut-être décrit par des besoins en ressources et des interactions entre tâches. On assigne des ressources de calcul et de mémoire (voire aussi des périphériques d'entrées-sorties) à chaque tâche, que ce soit des processus, threads ou tâches d'un graphe. L'interface MPI permet par exemple de réordonner les processus pour mieux les placer vis-à-vis du graphe de communication de l'application [71, 31]. De nombreux travaux ciblent actuellement ce domaine, en particulier la bibliothèque TreeMatch [38, 39] développée dans mon équipe et qui utilisent nos travaux sur la détection de la topologie.

2. On abordera la gestion mémoire pour architecture NUMA dans la section 5.2.

3. Les grappes de taille raisonnable sont organisées autour de 2 niveaux de switchs interconnectant des nœuds à deux processeurs multicœurs multithreads.

Une fois le placement des tâches décidé, il doit être appliqué pour effectivement verrouiller (*binding*) les tâches aux ressources qui leur ont été attribuées. Ce binding est un domaine beaucoup plus vaste qu'il n'y paraît car de nombreux composants logiciels sont impliqués. Tout d'abord, le gestionnaire de *batch* (qui permet aux utilisateurs de réserver des ressources matérielles) peut verrouiller les processus lancés à l'intérieur de ces ressources, il faudra alors affiner le binding sans contredire le gestionnaire. Ensuite une implémentation MPI peut elle aussi verrouiller les processus, soit au lancement (via `mpirun`), soit lors de l'initialisation (pendant `MPI_Init`). Une politique de placement de processus MPI devra donc être intégrée à ces techniques pour ne pas entrer en conflit avec elles. [4] Une application utilisant des bibliothèques parallèles comme les BLAS de la MKL d'Intel devra aussi veiller à ce que cette dernière soit capable de placer ses threads en respectant le binding global du processus. D'une manière générale, une forte collaboration de tous ces composants est indispensable pour que l'ensemble des tâches soit bien placé et que les affinités soient bien suivies. Mais le mélange des composants provenant de projets open-source et de logiciels industriels fermés rend cette tâche difficile.

Au-delà de la simple compréhension de la hiérarchie de ressources matérielles et du binding des tâches, les applications peuvent également avoir besoin de consulter des informations pour mieux comprendre les affinités matérielles. Par exemple, quels sont les caches partagés entre ces cœurs ? Quelle est la taille de ce cache ? Voire d'autres caractéristiques comme la taille de ligne de cache pour optimiser un noyau de calcul. L'étape suivante consiste à indiquer à l'application l'impact de ces caractéristiques sur ses performances. Par exemple, comment vont varier mes performances si deux threads partageant une certaine quantité de données ne partagent plus de cache L2 ? On reviendra sur ce point en section 4.4.1.

4.1.3 Une topologie de plus en plus compliquée ?

La raison évidente pour laquelle les topologies matérielles modernes évoluent est que le nombre de cœurs dans les processeurs évoluent. Les processeurs généralistes sur le marché des serveurs en 2014 disposent d'entre 4 et 12 cœurs chez Intel et même jusqu'à 16 cœurs chez AMD. Et la tendance ne va pas s'arrêter puisque des processeurs à centaines de cœurs sont attendus dans 5 ans, et des coprocesseurs spécialisés à plusieurs dizaines de cœurs sont déjà disponibles.

Par ailleurs, on s'attendait à ce que le nombre de processeurs par machine augmente également mais il s'est en fait stabilisé à 2, pour des raisons de rapport prix/performance. Si certains constructeurs comme SGI sont capables de construire des supercalculateurs à plusieurs centaines de processeurs en mémoire partagée [5], ils restent cantonnés à des usages très spécifiques.

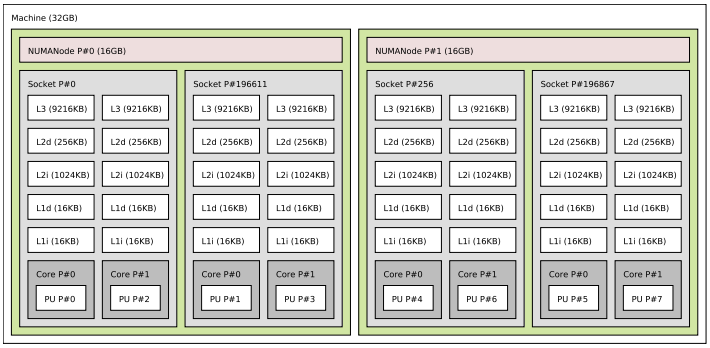
Ensuite, deux machines identiques matériellement peuvent être vues différemment par les applications car les ressources matérielles sont numérotées par le BIOS (voire par le système d'exploitation). Une mise à jour logicielle peut changer la numérotation et rendre caduque un placement de tâche précédemment optimisé.

La raison derrière ces numérotations variables est que les architectes des BIOS essaient de proposer une numérotation utile aux applications et systèmes d'exploitation qui ne savent pas s'adapter à la topologie. Comme ces applications vont utiliser prioritairement les *premières* ressources de calcul, les constructeurs vont tenter de numéroter en *premier* des ressources répondant bien aux besoins des applications. Cela peut se traduire par l'utilisation de cœurs de différents nœuds NUMA (pour maximiser la bande passante mémoire), l'utilisation d'un

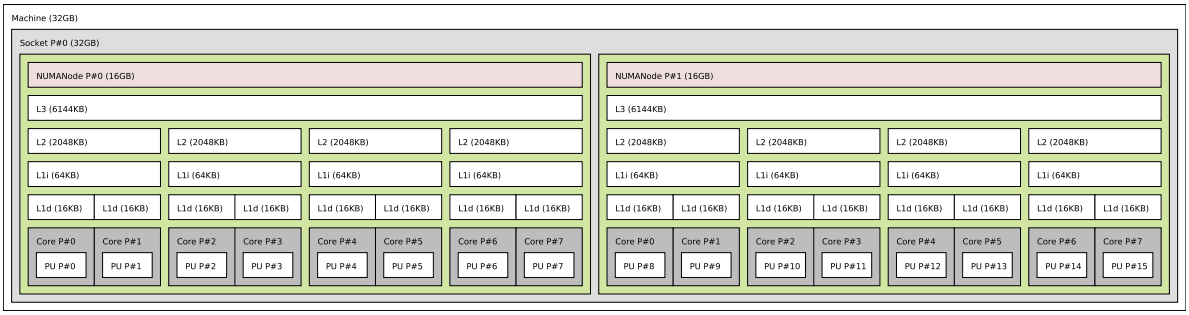
4. Une autre solution consiste à laisser l'implémentation MPI placer les processus et ensuite échanger leur rôle pour réorganiser les processus avec `MPI_Dist_graph_create` [37].

5. En 2013, l'Altix UV de SGI permettait déjà d'assembler 256 processeurs Intel Xeon E5, soit jusqu'à 3072 cœurs en mémoire partagée.

seul thread matériel par cœur (pour éviter que les autres threads ne le perturbent), etc. Par exemple, sur la figure 4.1(a) les unités d'exécutions (PU, *Processing Units*) sont numérotées horizontalement d'abord par processeur, puis par cœur, puis par nœud NUMA. Cependant, comme les besoins des applications varient beaucoup, les solutions choisies des constructeurs varient également beaucoup. La numérotation peut donc fortement varier d'une machine à l'autre.



(a) Une machine à quatre processeurs Intel Itanium 2 *Montecito* 9040 bicœurs. Chaque nœud NUMA contient deux sockets. Aucun cache n'est partagé entre cœurs.



(b) Un processeur AMD Opteron *Interlagos* 6276 hexadecacœur. Chaque socket contient deux nœuds NUMA. Les caches L3, L2 et L1i sont partagés entre certains cœurs, le L1d non.

FIGURE 4.1 – Exemples de topologies matérielles exportées par le logiciel `lstopo`.

L'ordre vertical ne peut également pas être présupposé constant, en particulier suite à l'introduction par AMD de topologies originales dans ses processeurs. Alors que les architectes placent habituellement un ou plusieurs processeurs par nœud NUMA (2 sur la figure 4.1(a)), les processeurs AMD haut de gamme contiennent chacun deux nœuds NUMA. En effet, comme montré sur la figure 4.1(b) chaque moitié du processeur est connecté directement à un banc mémoire, et moins rapidement à l'autre. Les nombreuses applications présupposant que le niveau NUMA est au dessus du niveau processeur dans la hiérarchie ne pourront donc pas gérer correctement ces machines.

De la même façon, les caches L1 d'instructions et données sont en général spécifiques à chaque cœur, sauf sur les derniers processeurs AMD où le cache L1i (L1 réservé aux instructions) est partagé entre paire de cœurs, comme le L2. Évidemment, on ne peut pas non plus présupposer l'existence ou l'absence de certains niveaux puisque les niveaux de cache varient selon le type de processeur.

Pour toutes ces raisons, il est difficile pour une application de détecter la topologie matérielle en restant suffisamment générique pour supporter toutes les architectures modernes. Pour exemple, le support OpenMP du compilateur Intel (bien placé pour maîtriser les archi-

tectures x86) ne gère en fait pas correctement tous les cas. [6] C'est la raison pour laquelle nous avons développé le logiciel hwloc.

4.2 Décrire la topologie matérielle avec hwloc

Je décris dans cette section comment notre logiciel hwloc (*Hardware Locality* [7]) est devenu la bibliothèque incontournable pour la consultation de topologie et le placement des tâches, puis j'en explique les concepts.

4.2.1 Du placement des threads au placement des processus, l'histoire d'hwloc

Les bases de hwloc sont nées pendant la thèse de Samuel Thibault qui s'intéressait à l'ordonnancement des threads sur machines hiérarchiques [68, 69]. Il avait développé des outils découvrant les nœuds NUMA, processeurs, cœurs et threads matériels. Ils étaient portables sur de nombreux systèmes d'exploitation mais imposaient des contraintes sur l'ordre des niveaux de hiérarchie et ignoraient certains niveaux, ce qui allait à l'encontre des besoins listés dans la section précédente.

J'ai modernisé l'implémentation en utilisant de nouvelles sources d'information de topologie, par exemple les fichiers virtuels `sysfs` sous Linux, et ajouté la détection des caches. Chose a priori anecdotique, j'ai également écrit l'outil qui affiche la topologie de la machine. Mais le succès de son descendant `lstopo` (qui génère automatiquement des sorties textuelles ou graphiques telles que les figures 4.1) a en fait montré qu'un tel outil était un levier de conviction des utilisateurs car il démontre en un clin d'oeil la connaissance poussée de la topologie dont nous disposons.

Ces petites avancées ont attiré l'attention de mes collègues qui commençaient à travailler sur le placement de processus MPI et avaient donc besoin d'information de topologie eux aussi. Nous avons donc extrait la détection de topologie du logiciel Marcel où elle était intégrée pour en faire une bibliothèque séparée, `libtopology`. Ce travail a été ensuite remarqué par les développeurs d'Open MPI qui disposaient d'un projet concurrent (PLPA, *Portable Linux Processor Affinity*) dont ils commençaient à voir des limites critiques (uniquement sous Linux, nœuds NUMA et caches ignorés). Nous avons fusionné les deux projets en hwloc en utilisant notre code et la popularité d'Open MPI. Depuis hwloc a également été intégré à MPICH et s'est répandu dans de nombreux autres logiciels, en particulier ceux qui doivent placer des tâches de calcul pour le HPC. [8]

4.2.2 Organisation logique des ressources

Si on met de côté l'avantage marketing de l'outil `lstopo` pour convaincre les utilisateurs, le succès de hwloc est dû à la connaissance avancée du matériel et la façon dont il l'expose.

Compte-tenu des contraintes listées en section 4.1.3, nous avons conçu hwloc pour représenter la topologie sous la forme d'un arbre, comme schématisé sur la figure 4.2. Les ressources de calcul et mémoire sont organisées en arbre selon un ordre d'inclusion. Tous les objets de même type (tous les nœuds NUMA, tous les processeurs, tous les caches L1i...) sont placés sur un même niveau horizontal (et reliés), ce qui permet de les parcourir facilement, même si la topologie est asymétrique (si un processeur contient moins de niveaux de cache qu'un autre).

6. ICC ne détecte pas correctement la topologie de certaines machines à processeurs AMD, ou à processeurs Intel interconnectés par des chipsets IBM spécifiques.
7. hwloc est disponible sous licence BSD à `http://www.open-mpi.org/projects/hwloc/`.
8. Une liste des logiciels utilisant hwloc est disponible sur le site web du projet.

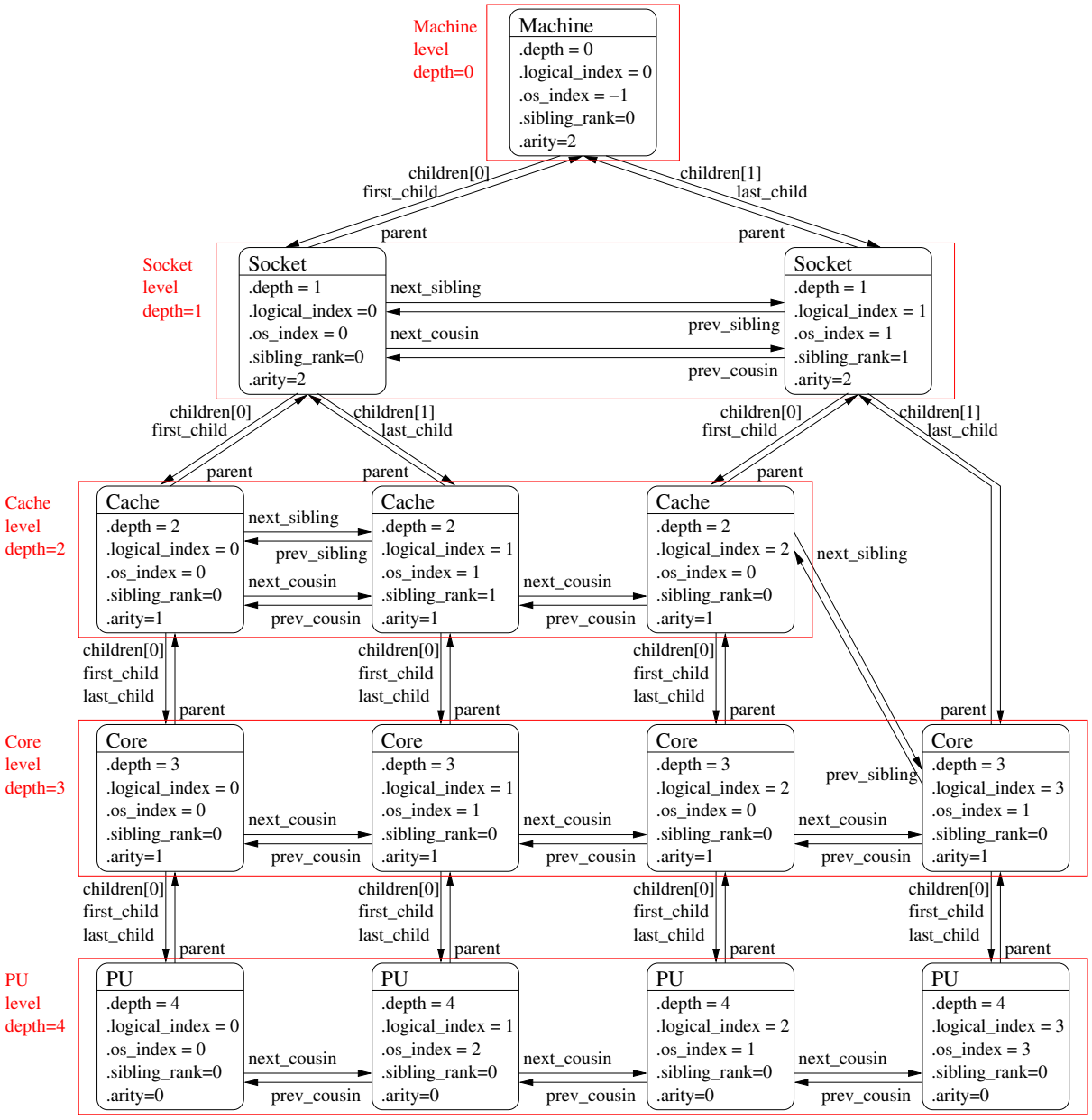


FIGURE 4.2 – Organisation d'une topologie hwloc sous la forme d'arbre.

Aucun ordre n'est imposé entre les niveaux, ce qui permet d'éviter les problème cités précédemment avec les sockets et les nœuds NUMA. L'application peut simplement demander où se trouve le niveau NUMA, ou quel type d'objet se trouve au dessus ou au dessous des sockets.

Ces niveaux servent par ailleurs à renuméroter les objets de manière logique. Les cœurs dont les numéros logiques se suivent seront donc physiquement proches, contrairement aux numérotations compliquées et variables citées en section 4.1.3. Ces numéros logiques permettent ensuite à l'application de placer ses tâches facilement. Chaque objet dans l'arbre contient par ailleurs des attributs spécifiques comme par exemple la taille et l'associativité s'il s'agit d'un cache. La modélisation en arbre est donc simple mais n'interdit pas à l'application d'utiliser des informations bas-niveau sur le matériel.

Si la numérotation logique des objets est une force pour hwloc (car elle masque définitivement les renumérotations que les mises à jour du BIOS ou du système d'exploitation peuvent engendrer), elle est aussi un frein à son adoption. En effet, les problèmes de numérotation se sont généralisés depuis une petite dizaine d'années, et les experts se sont habitués à les contourner manuellement, ce qu'ils doivent maintenant oublier en faisant confiance à un outil développé par d'autres.

La principale critique faire à notre modélisation en arbre est qu'elle ignore l'interconnexion à l'intérieur de chacun des niveaux. Les cœurs à l'intérieur d'un processeur sont par exemple parfois reliés en anneau. Le réseau mémoire entre les nœuds NUMA est quant à lui parfois très particulier compte tenu du nombre limité de liens disponibles sur les processeurs (on y reviendra en section 5.1.1). S'il est vrai que cet aspect n'apparaît pas sur l'arbre, rien n'empêche de le stocker à côté. hwloc peut donc annoter les niveaux avec des informations sur la distance entre les différents objets, ce qui permet par exemple de savoir si deux nœuds NUMA sont physiquement plus proches que d'autres. La structure d'arbre représente donc la localité des objets en ce qui concerne l'inclusion, tandis que ces annotations représentent les distances entre objets de même type.

4.2.3 Au-delà des simples processeurs

Si la capacité d'hwloc à placer des tâches sur certains processeurs (ou des données sur certains bancs mémoire) ne peut pas être étendue, l'interface de consultation de la topologie peut être généralisée à d'autres informations de localité. J'ai tout d'abord ajouté à hwloc la connaissance des bus d'entrées-sorties (PCI), ce qui permet notamment de savoir quels périphériques sont proches de quels cœurs. Non seulement les périphériques matériels y sont présentés, mais également les périphériques logiciels, comme par exemple une interface `eth0`, une partition `sda` ou une connexion InfiniBand. Cela permet aux applications de connaître rapidement la localité des descripteurs logiciels de périphériques qu'elles manipulent [CI6]. Nous présenterons l'utilisation de ces fonctionnalités dans la section 5.3.

L'arbre de hwloc peut également être prolongé vers le haut pour décrire la topologie d'un réseau hiérarchique de machines (et switchs). L'idée est de regrouper toutes les informations de localité d'un système dans une seul et unique interface, que ce soit pour une seule machine, une grappe ou un supercalculateur [CI6]. J'ai ajouté à hwloc la possibilité d'assembler les topologies de différents nœuds selon l'organisation réseau. Cela permet aux utilisateurs d'appliquer des algorithmes globaux, à la fois entre et à l'intérieur des nœuds.

Cette idée a cependant deux inconvénients majeurs : d'une part la vue en arbre est peu adaptée à la topologie réseau car la notion d'inclusion a peu de sens quand l'interconnexion est complexe comme par exemple dans un réseau en tore à 3 dimensions. D'autre part, le vaste problème de la détection de topologie réseau, et la variété des technologies réseau existantes, est un frein à l'intégration de ce support directement dans hwloc. Une collaboration avec Cisco

et l'Université du Wisconsin à La Crosse cible actuellement ces problèmes en tentant de combiner hwloc, pour la connaissance de l'intérieur de nœuds, avec un outil dédié aux réseaux, notamment InfiniBand et Ethernet, sous forme de graphes quelconques et non plus d'arbres. Ce travail se traduit par le développement du logiciel netloc [9].

Une autre voie d'extension d'hwloc que nous explorons, cette fois vers le bas de l'arbre, consiste également à décrire l'intérieur des accélérateurs (GPGPU, Xeon Phi...).

Toutes ces fonctionnalités permettant à hwloc d'avoir une connaissance de plus en plus poussée du matérielle a cependant un effet pervers : les utilisateurs souhaitent souvent étendre `lstopo` pour avoir un affichage plus fin de leur machine, sans apporter aucun intérêt technique ou scientifique pour les applications.

4.2.4 Où s'arrête le système d'exploitation ?

Le développement d'hwloc pose la question de la séparation entre le système d'exploitation, son noyau, et les bibliothèques. Beaucoup de bibliothèques HPC sont considérées comme des extensions naturelles du système, et c'est encore plus vrai pour hwloc. En effet, le système est censé masquer tous les détails techniques non portable du matériel et exposer des informations simples aux applications. Mais les applications HPC souhaitent de plus en plus souvent connaître le matériel pour mieux s'y adapter. hwloc souhaite disposer de beaucoup d'informations matérielles afin de les réorganiser et abstraire de façon plus adaptée au HPC, mieux que ce que les systèmes d'exploitation savent faire.

Il existe plusieurs façons de détecter la topologie. La première consiste à accéder directement au matériel, quand c'est possible. Les processeurs modernes disposent d'instructions spéciales permettant de connaître tous leurs détails. [10]. Mais comprendre le résultat de ces instructions impose une veille technologique permanente afin de supporter tous les nouveaux modèles de processeurs et tous les cas particuliers qu'ils amènent. hwloc n'utilise cette technique que lorsque le système d'exploitation ne fournit pas directement d'information de topologie (par exemple sous FreeBSD).

L'autre solution naturelle consiste à demander des informations au système d'exploitation. Les noyaux Linux et Solaris fournissent par exemple beaucoup d'informations de topologie. Les constructeurs veillent à ajouter le support des nouveaux processeurs très en amont dans les systèmes d'exploitation principaux, afin que le code soit disponible dans les distributions quand le matériel est disponible. hwloc n'a alors besoin que d'une distribution récente pour espérer avoir des informations de topologie à jour. [11] Là où le bât blesse c'est dans la quantité d'information que le noyau expose effectivement aux applications. Les développeurs noyau ont parfois tendance à ne pas voir l'intérêt d'en exposer beaucoup car ils pensent que seul le noyau en a besoin. [12] On reviendra sur la collaboration entre recherche, HPC et développeurs du noyau Linux en conclusion 6.3.2.

Une dernière façon de détecter la topologie consiste à mesurer les distances entre les différents objets pour en déduire leur organisation. hwloc utilise cette technique pour préciser le résultat des autres stratégies, nous y reviendrons en section 4.4.1.

9. netloc est disponible sous licence BSD à `http://www.open-mpi.org/projects/netloc/`.
10. Par exemple `cpuid` sur architectures x86.
11. Et les bugs hwloc se traduisent parfois en bugs du noyau Linux quand certaines spécificités des caches AMD sont oubliées, `http://bugzilla.kernel.org/show_bug.cgi?id=42607`.
12. Pour Ingo Molnar, l'important est d'exposer les informations de topologie à l'ordonnanceur Linux et éventuellement aux allocations des pilotes de périphériques, mais pas aux applications, `http://lkml.org/lkml/2009/6/7/148`.

4.2.5 Un succès dans le HPC voire au-delà

hwloc rencontre un franc succès depuis sa première révision en 2009. Il est désormais distribué avec la plupart des implémentations MPI mais aussi des supports exécutifs, des gestionnaires de batchs et des bibliothèques numériques. Nous avons présentées quelques applications directes d'hwloc dans le contexte du HPC dans l'article [CI1]. Cela montre comment le logiciel s'est imposé à la racine de tous les travaux récents de mon équipe de recherche, que ce soit sur les communications intra- ou inter-nœud ou sur l'ordonnancement de threads ou de tâches.

Le logiciel commence également à s'imposer dans le monde des administrateurs système car il représente rapidement l'ensemble des ressources matérielles d'une machine [RA2]. De plus des outils tels que `hwloc-ps` permettent d'étendre les fonctions de monitoring habituelles en ajoutant des informations de topologie.

4.3 Modéliser les accès mémoire

hwloc ne fournit essentiellement que des informations qualitatives, c'est-à-dire une organisation des ressources en arbre. Les applications peuvent savoir si deux cœurs sont proches d'un même banc mémoire ou partagent un cache. Mais elles ne peuvent pas savoir si le respect de cette localité est critique pour la performance. Par exemple la présence d'un cache partagé joue habituellement sur les performances d'accès à de la mémoire partagée. Mais qu'en est-il du placement de deux threads dans un même processeur multicœur ou dans deux processeurs différents ? Cela dépend notamment des contentions sur le lien reliant le processeur à la mémoire.

Il s'agit donc de quantifier comment les différents niveaux de ressources sont imbriqués et/ou indépendants. C'est un problème difficile qui impose de modéliser des accès mémoire de l'application et leur implémentation dans le matériel. C'est ce sur quoi nous avons travaillé avec Bertrand Putigny pendant sa thèse [58].

4.3.1 Des architectures mémoire très complexes

Alors que les processeurs ont vu leur puissance augmenter énormément depuis vingt ans, le fossé avec la vitesse de la mémoire s'est agrandi. On sait construire des mémoires très rapides, par exemple les caches L1 des processeurs peuvent lire ou écrire une donnée en une nanoseconde. Mais on est incapable de conserver cette vitesse sur de gros volumes mémoire où le temps d'accès stagne vers 100 ns.

Les architectes matériels ont donc développé de nombreuses techniques pour masquer cette latence, en particulier les caches qui exploitent la localité temporelle et spatiale pour accélérer les accès aux données les plus souvent utilisées. Mais aussi, le *prefetcher* qui anticipe des accès mémoire pour éviter d'avoir à les attendre quand ils seront réellement nécessaires, et l'exécution dans le désordre (*out-of-order*) qui masque la latence des accès mémoire en exécutant d'autres instructions.

Toutes ces idées font qu'un simple programme va avoir un comportement mémoire difficilement explicitable. La puissance observée par un programme est devenue totalement différente de la puissance théorique annoncée par les constructeurs de processeurs, comme nous l'avons expliqué dans l'article de vulgarisation scientifique [MS4].

Si les instructions de calcul ont des temps d'exécution bien maîtrisés, les instructions accédant à la mémoire sont beaucoup moins prévisibles, rendant la modélisation des performances difficiles. De plus, les constructeurs diffusent peu d'informations sur l'implémentation de ces

instructions. On peut mesurer leur temps d'exécution expérimentalement [25] mais cela se résume à l'accès au cache de plus bas niveau, pas à la mémoire centrale.

4.3.2 Beaucoup trop de paramètres

Modéliser les accès mémoire pour comprendre le comportement des codes parallèles selon les affinités logicielles et matérielles impose de prendre en compte les fonctionnalités présentées à la section précédente. La présence des caches est d'ailleurs beaucoup plus complexe qu'il n'y paraît car dans le contexte multicœur et/ou multiprocesseur, ils imposent un protocole de cohérence pour maintenir à jour les différentes copies des données dans les différents caches. Ce protocole a un coût car il impose des transferts et invalidations entre les différents caches. La bande passante mémoire théorique (assez bien connue) peut donc s'effondrer si beaucoup de mémoire est partagée entre tâches car le maintien de la cohérence peut être fortement sollicité. On observe par exemple sur la figure 4.3(a) que l'écriture dans une ligne de cache est plus lente si des copies de cette ligne existent dans d'autres caches (état Shared).

Les processeurs actuels reposent sur des variantes du protocole MESI [55] qui explicite le comportement de ces différentes copies. On peut donc envisager de modéliser les accès mémoire en simulant le comportement des caches [33]. Mais cela impose de prendre en compte ce protocole pour chacune des lignes de cache (des centaines de milliers dans un serveur moderne), ce qui conduit à des simulateurs extrêmement lents. Par ailleurs, il faut connaître des détails internes de l'architecture, tels que les temps d'accès entre les différents niveaux de cache, ou la façon dont sont implémentées les différentes transitions d'état. Par exemple quel est le coût de l'invalidation des copies distantes lors d'un passage de l'état Shared à Modified ? Les constructeurs ne diffusent pas ces détails et cette sous-étape ne peut pas être mesurée individuellement.

4.3.3 Utiliser des micro-benchmarks pour cacher la complexité

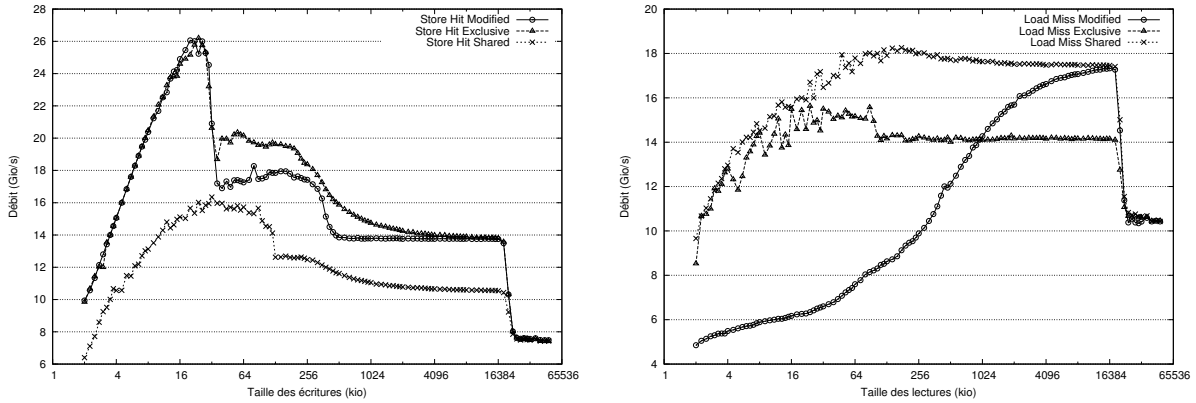
Comme il est apparu que l'architecture mémoire est trop compliquée pour être simulée précisément en temps raisonnable, nous avons décidé de nous concentrer sur l'aspect qui nous paraît le plus important : la cohérence de cache. Nous pensons que le coût du maintien de la cohérence est le principal facteur permettant de décider si l'affinité entre des tâches est critique pour la performance. L'objectif de nos travaux sur la topologie des machines est plus de comprendre les variations de performance selon le déploiement d'un code (combien de cœurs il utilise et lesquels) que sa performance absolue. On peut donc ignorer les fonctionnalités matérielles qui importent peu pour notre étude, par exemple l'out-of-order ou le prefetching.

L'idée de la thèse de Bertrand Putigny a consisté à les cacher dans des micro-benchmarks *boîte-noire*. On représente un code par une suite d'accès à des blocs mémoire qui font transiter ces blocs entre les différents états de cache. On mesure alors les performances des accès selon l'état du cache et sa taille, puis on combine ces mesures pour obtenir la performance globale. Ce modèle cible essentiellement les codes *memory-bound* où les accès mémoire sont les facteurs limitants. Comme expliqué plus haut, les codes *compute-bound* sont moins difficiles à simuler et leur performance varie peu avec le placement et la topologie matérielle.

Il faut cependant que le code original et nos micro-benchmarks utilisent aussi efficacement l'architecture. Le code original est normalement généré par des compilateurs performants et/ou il utilise des bibliothèques de calcul performantes comme la MKL d'Intel. Nos micro-benchmarks sont générés en assembleur par un outil développé par Bertrand qui vise à optimiser au mieux les accès mémoire pour mesurer précisément la bande passante crête [13].

13. L'outil *mbench* est disponible en ligne à `https://github.com/bputigny/mbench`.

Cet outil va par ailleurs faire le nécessaire pour placer les blocs mémoire dans les états de cache désirés pour chaque micro-benchmark. Il permet donc de mesurer les débits mémoire dans les différents états possibles des caches, en considérant le cache du processeur local (celui qui fait l'accès) et ceux des cœurs voisins.



(a) Écriture dans zone cachée localement.

(b) Lecture dans zone cachée par autre cœur.

FIGURE 4.3 – Performance des accès mémoire selon les états de cache dans protocole de cohérence MESI. La plate-forme est constituée de 2 processeurs octocœurs Xeon *Sandy-Bridge* E5-2650.

La figure 4.3 montre certains résultats de ces micro-benchmarks. Elle illustre aussi le fait que la taille des caches est intégrée à ces courbes (les 20 Mo du cache L3 créent une chute de performance) et n'aura donc pas à être explicitement considérée dans le modèle mathématique plus tard. En effet, si un bloc ne tient pas dans le cache, le micro-benchmark mesurera la vitesse d'accès à la mémoire centrale.

La figure montre aussi que l'état d'une ligne de cache influe significativement sur les performances des défauts de cache. Cela remet par exemple en question l'idée très répandue d'évaluer les performances d'une application en observant simplement son nombre de défauts de cache.

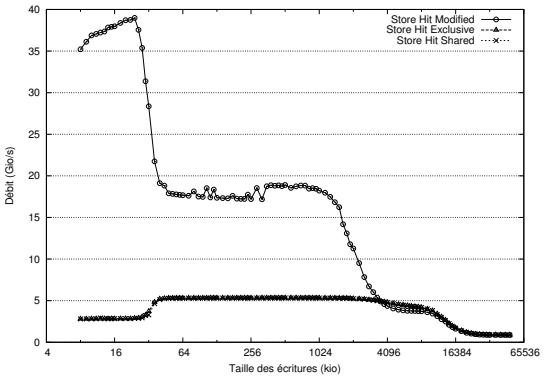


FIGURE 4.4 – Performance des écritures selon l'état de la zone destination dans le cache local dans protocole de cohérence MESI. La plate-forme est constituée de 16 processeurs hexacœurs Xeon *Dunnington* X7460.

L'outil a d'ailleurs permis à Bertrand d'expliquer les faibles performances mémoire obser-

vées sur notre plate-forme à 96 cœurs en mémoire partagée[14]. La figure 4.4 montre qu'écrire dans une zone en Exclusive est aussi lent qu'en Shared, alors que l'état Exclusive a justement été créé pour améliorer ce cas en évitant les messages de cohérence ! Cela a révélé un problème dans l'implémentation du protocole MESI dans le processeur et/ou le contrôleur mémoire, et a fait regretter le choix de cette plate-forme comme prototype à plusieurs dizaines de cœurs...

4.3.4 Modéliser les codes parallèles en combinant des micro-benchmarks

Une fois le comportement de l'architecture mémoire extrait sous forme d'un ensemble de micro-benchmarks, nous les combinons pour prédire les performances des noyaux de calcul. La méthode est décrite en détail dans [CI15], [58]. Les codes considérés étant memory-bound, les instructions ne touchant pas à la mémoire peuvent être ignorées. On ne conserve donc que les accès mémoire, comme illustré par les figures 4.5(a) et 4.5(b). Les dépendances entre instructions peuvent être généralement ignorées car elles ne concernent qu'un élément mais disparaissent une fois que les accès globaux aux blocs de données sont pipelinés.

```
#pragma omp parallel for private(i)
for (i=0; i<SIZE; i++)
    Y[i] = a * X[i] + Y[i];
```

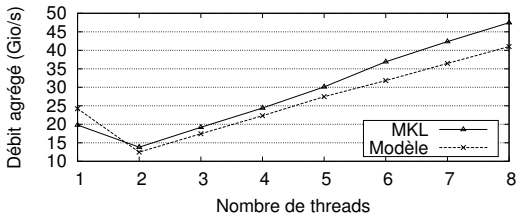
(a) Code original parallélisé avec OpenMP.

$$read_{f(p)=X_p}$$
$$read_{f(p)=Y_p}$$
$$write_{f(p)=Y_p}$$

(b) Modélisation des accès mémoire.

$$T_{daxpy} = \frac{size}{lm(size,M,n)} + MAX\left\{\frac{size}{lm(size,M,n)}, \frac{size}{sh(size,S,n)}\right\}$$

(c) Calcul du temps d'exécution du modèle.



(d) Temps d'exécution observé (MKL) et prédit, pour un vecteur de 32 ko, selon le nombre de threads.

FIGURE 4.5 – Modélisation d'un noyau DAXPY.

Un automate peut alors considérer les blocs mémoire et traiter chacun des accès. À chaque étape, on ajoute le temps d'exécution prédit par les micro-benchmarks et on met à jour l'état des blocs dans les caches. L'arithmétique de calcul des temps d'exécution tient compte de la capacité des processeurs modernes à effectuer une lecture et une écriture en même temps, mais pas deux lectures ou deux écritures. On combine donc les temps de deux benchmarks par addition ou par prise du maximum selon les cas, comme illustré sur la figure 4.5(c).

Ce modèle a été appliqué à différents noyaux de calcul memory-bound répandus (dot-product, daxpy, FFT, gradient conjugué) et permet de prédire assez bien l'évolution des performances selon la taille des données ou selon le nombre de threads [CI15], [58]. L'exemple détaillé sur la figure 4.5(d) permet également de voir que ce noyau de calcul souffre beaucoup du maintien de la cohérence (passer d'un seul thread à deux dégrade significativement les performances) et imposera donc un placement proche de ces threads.

Ce travail impose cependant de disposer des bons micro-benchmarks représentatifs des accès mémoire du code à modéliser. Si Bertrand a pu développer un outil générant des codes

14. IBM x3950M2 assemblant 4 nœuds à 4 processeurs Xeon *Dunnington* X7460.

fiables (utilisant les bons états de cache) et reproductibles, il reste à clairement définir lesquels sont nécessaires. En effet, quand les processeurs deviennent de plus en plus multicœurs, le nombre de choix de cœurs utilisés explose combinatoirement. À défaut d'exécuter un micro-benchmark pour chaque combinaison de cœurs, il faut envisager de prédire les comportements des cas complexes à partir des cas séquentiels ou parallèles simples.

4.4 Bilan et perspectives

L'augmentation de la complexité des machines est aujourd'hui incontournable. Les applications doivent en tenir compte avant d'espérer exploiter la puissance de calcul de manière satisfaisante. J'ai attaqué ce problème de deux façons. D'une part le logiciel hwloc permet de modéliser l'architecture en un arbre de ressources organisées selon leur localité afin d'exposer des informations portables aux applications et de leur faciliter les opérations de placement. D'autre part, nous avons proposé une modélisation des codes parallèles memory-bound basée sur des combinaisons de micro-benchmarks qui donne des indications sur le comportement selon la taille des données et le degré de parallélisme.

4.4.1 La nécessité d'informations quantitatives

Une modélisation précise du comportement de la machine et des codes parallèles devient indispensable à la prise de bonnes décisions d'ordonnancement, répartition des tâches, etc. Cela s'applique évidemment à l'optimisation des performances, mais aussi à d'autres critères comme l'amélioration de la consommation énergétique. Quand les constructeurs fournissent des modèles fiables, l'application peut s'en servir directement pour s'adapter au matériel. C'est par exemple ce que nous avons fait sur le processeur expérimental SCC d'Intel pour trouver un compromis entre puissance de calcul et consommation d'énergie [WI9].

Malheureusement, dans la plupart des cas, les constructeurs eux-mêmes sont incapables de donner des informations quantitatives car ils ne maîtrisent plus le comportement global des architectures qu'ils conçoivent. Par exemple, le *TurboBoost* d'Intel modifie la performance séquentielle d'un cœur en jouant sur la consommation énergétique globale du processeur sans contrôle logiciel. Le travail de nos outils logiciels est alors de trouver d'autres moyens d'aider à la prise de décision, avec des critères chiffrés. hwloc peut d'ores et déjà fournir certaines informations quantitatives, par exemple le débit mémoire entre un cœur et un nœud NUMA ou la latence entre un cœur et un cache. Ces informations doivent être mesurées expérimentalement et ajoutées à la topologie hwloc pour être ensuite exposées aux applications. La topologie peut même être précisée en fonction de ces mesures [28] [15].

Nous étudions par exemple cette stratégie dans le projet SEHLOC [16] pour paramétriser les performances de communications dans le modèle Multi-BSP (version *Multicore* et hiérarchique du modèle *Bulk Synchronous Parallelism* [72]) puis prédire les performances de certaines applications parallèles. Mais ces informations restent simples et ne peuvent pas toujours être directement traduites en critères de décision, qui dépendent de l'application, et en particulier du volume de données partagées. Si la thèse de Bertrand Putigny va dans ce sens, il reste du chemin avant de faire remonter les résultats de telles modélisations jusqu'au niveau de l'application. Nous étendons également ce travail pour configurer automatiquement les seuils de transition entre les stratégies de communication intra-nœud dans les implémentations MPI

15. hwloc peut analyser des matrices de distances fournies par le BIOS ou l'utilisateur pour créer des groupes virtuels d'objets proches, par exemple les nœuds NUMA voisins dans une grande machine.

16. Projet STIC-AmSud en collaboration avec l'Universidad Nacional de San Luis (Argentine) et l'Universidad de la República (Uruguay). http://runtime.bordeaux.inria.fr/sehloc/.

comme évoqué dans le chapitre 3. Le stagiaire Benoit Ruelle a travaillé à la modélisation de ces transferts de données particuliers pour mieux comprendre le comportement de ces stratégies [WI10] et nous avons pu en déduire des idées d'optimisation pour les développeurs d'implémentation MPI. Et on reviendra sur d'autres problématiques de choix de stratégie dans la section 5.2 à propos de la question de migrer des threads et/ou de la mémoire, ou de conserver des accès mémoire lents.

Une autre solution étudiée dans notre équipe dans le logiciel StarPU [7] consiste à utiliser un historique des exécutions précédentes pour prédire les temps d'exécution futurs. Cette idée permet une adaptation automatique de l'application aux performances des différentes unités de calcul grâce aux performances des exécutions des codes précédents. Cependant, elle ne s'applique qu'aux codes dont les performances ne dépendent pas des données en entrées, en particulier les noyaux d'algèbre linéaire dense. D'autre part, avec l'augmentation du nombre de cœurs, bancs mémoire, accélérateurs, etc., il va falloir comparer de nombreuses configurations d'exécution possibles, par exemple le nombre de cœurs à utiliser pour un noyau, lesquels, et où déplacer les différentes données. Il faudra voir si on continue à pouvoir prendre de bonnes décisions en temps raisonnable, c'est-à-dire beaucoup plus rapidement que le temps de l'exécution elle-même.

4.4.2 Ne plus imposer une connaissance des machines cibles

L'intégration de hwloc dans Open MPI et MPICH a permis l'ajout de fonctionnalités avancées de placement de processus. Les gestionnaires de batch comme Torque peuvent également tenir compte d'informations de topologies pour l'allocation de ressources. Cependant, il reste une limite forte à ces solutions : l'utilisateur doit connaître la machine pour configurer ces allocations. Il devra explicitement indiquer combien de ressources de chaque niveau doivent être utilisées. Par exemple, si 4 machines à 8 cœurs sont disponibles pour exécuter 16 processus gourmands en bande passante mémoire, l'utilisateur doit expliciter qu'il veut 4 processus par machine. Dans une grappe hétérogène, on ne peut pas préciser ainsi avant de connaître les machines allouées, ce qui empêche d'effectuer l'allocation de ressources et le placement simultanément.

De nombreux outils de placement ne sont toujours pas capables de répartir intelligemment les tâches dont on aurait spécifié les besoins indépendamment de la topologie sous-jacente. Ce problème a été attaqué par le *Locality Aware Mapping Algorithm* récemment intégré dans Open MPI qui permet de choisir l'ordre dans lequel différents niveaux hiérarchiques seront considérés lors de la répartition des processus [34]. L'exemple précédent pourra alors être traité de manière générique en spécifiant qu'on veut répartir d'abord par machine puis par cœur. Cette approche présente l'avantage d'éviter à l'utilisateur d'avoir à connaître la taille des différents niveaux (combien de cœurs par cache L2, combien de sockets par nœud...). Mais il doit toujours connaître les différents niveaux hiérarchiques existants et leur impact sur les performances. En effet, on ne répartit pas de la même façon des processus qui se synchronisent souvent par paire sur une machine à cache partagé par paire ou sans cache partagé.

Il faut maintenant aller vers un modèle de plus haut niveau permettant à l'utilisateur de décrire ses besoins en termes totalement indépendants du matériel. L'idée est assez similaire aux arbres de bulles pour l'ordonnancement des threads dans la thèse de Samuel Thibault [68] mais appliquée ici à d'autres outils de placement, en particulier les batch-schedulers et les lanceurs d'application, par exemple MPI. Idéalement, l'utilisateur spécifierait que son application est composée d'un arbre de sous-groupes avec différents besoins, et un algorithme les placerait sur les niveaux les plus adaptés (nœud NUMA pour la bande passante mémoire, gros cache pour le partage mémoire ou les communications, petit cache rapide pour la synchronisation...).

C'est en quelque sorte une généralisation de TreeMatch à différentes métriques d'optimisation. La nécessité de quantifier ces besoins et les paramètres des différents niveaux hiérarchiques ressurgit alors. Et les travaux comme la thèse de Bertrand Putigny pourraient être un moyen d'y répondre.

Cependant, il ne faut pas totalement cacher la topologie matérielle car l'application va en avoir besoin pour s'optimiser. Si les algorithmes *cache-oblivious* [26] sont capables d'exploiter efficacement un processeur sans connaître la taille des caches, de nombreux problèmes ne peuvent pas en faire autant. Si le développeur doit tenir compte directement des caractéristiques des caches lors de la programmation de son application, pourquoi n'en tiendrait-il pas compte lors de son placement ?

Chapitre 5
Optimiser les mouvements de données dans les machines NUMA

J'ai présenté dans le chapitre 4 notre outil hwloc pour gérer la localité des ressources. Il permet d'obtenir des informations matérielles pour prendre des décisions et d'appliquer facilement un placement des tâches et des données. La non-uniformité des architectures (NUMA) s'est répandue dans les serveurs modernes, apportant de nouvelles contraintes car les problèmes de la localité s'appliquent aussi aux données manipulées, pas seulement à l'exécution du code. J'ai notamment présenté tous ces problèmes de localité mémoire dans l'article de vulgarisation scientifique [MS2].

Si hwloc sait nous fournir des informations sur cette localité mémoire, il faut également implémenter des moyens de déplacer les données entre les différents composants, et ce problème est beaucoup plus vaste que celui du placement des tâches. Je vais d'abord présenter l'état actuel des architectures NUMA et leurs évolutions, puis je présenterai nos travaux dans le cadre de l'ordonnancement des threads OpenMP et des communications MPI. Ces résultats visent à faciliter l'exploitation des architectures mémoire de plus en plus complexes et entrent donc dans mon second grand axe de recherche présenté en section 1.3.2.

5.1 NUMA et plus si affinités

Les architectures NUMA existent depuis plus d'une vingtaine d'années. Elles étaient initialement utilisées essentiellement pour la programmation parallèle en mémoire partagée à des degrés de parallélisme que les multiprocesseurs classiques ne pouvaient supporter. Les recherches ont rapidement mis en évidence l'importance du placement des tâches [12] et des données [1] et ces problèmes restent d'actualité.

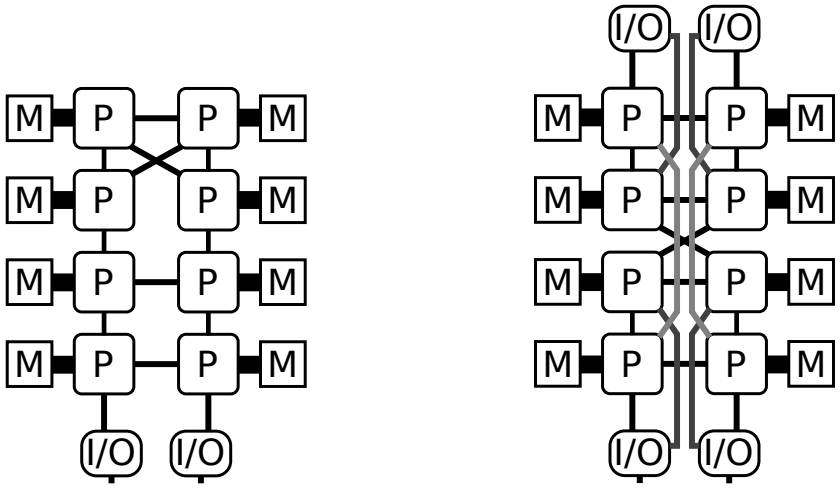
Cependant, les architectures NUMA sont désormais présentes dans la très grande majorité des serveurs car les fortes contraintes sur le bus mémoire se sont généralisées avec l'arrivée des processeurs multicœurs. Les problèmes de localité mémoire s'appliquent donc non seulement aux modèles de programmation par mémoire partagée, tel que OpenMP, mais à tous les autres, comme par exemple MPI.

5.1.1 Interconnexion mémoire à complexité variable

Les premières architectures NUMA consistaient en un assemblage de machines UMA (*Uniform Memory Access*). Un réseau d'interconnexion spécifique (et onéreux) était utilisé pour relier différents bus mémoire. Par exemple, la machine *Hector* [74] utilisait des modules composés d'un processeur et de mémoire sur un bus traditionnel, avant que plusieurs modules soient reliés par un anneau local, lui-même relié à l'autres par un anneau global. Le bus mémoire

classique n'avait donc pas à interconnecter directement de nombreux processeurs, chose pour laquelle il n'avait pas été conçu. La machine *Alewife* assemblait quant à elle des modules processeur/mémoire dans une grille 3D [2]. La distinction est alors très claire entre les nœuds UMA et leur interconnexion externe NUMA.

Le NUMA s'est démocratisé à partir de 2003 avec le processeur AMD Opteron et son bus mémoire HyperTransport. L'intégration du contrôleur mémoire dans ces processeurs a gommé la distinction entre les niveaux UMA et NUMA historiques. Au lieu d'une interconnexion NUMA ad-hoc, l'aspect distribué de la mémoire devenait une partie intégrante des processeurs. Chaque banc mémoire est désormais connecté de manière privilégiée à un seul processeur, et accessible moins rapidement par les autres.



(a) Serveur à 8 processeurs AMD Opteron 800 ou 8000 (2004-2010).

(b) Serveur à 8 processeurs Intel Xeon 7500 ou E7-8800 (2010-2012).

FIGURE 5.1 – Topologie NUMA de plates-formes multiprocesseurs généralistes. P désigne les processeurs, M la mémoire, et I/O les bus d'entrées-sorties.

Cette intégration et l'impossibilité d'être très extensible à prix abordable a conduit chaque génération de processeurs à imposer de fortes contraintes en terme d'interconnexion. Les processeurs Opterons ne pouvaient par exemple être placés que dans des machines à 8 processeurs au mieux, et chacun d'eux ne disposait que de 3 liens directs vers ses voisins. Cela a donné lieu à des configurations mémoire très particulières comme montré sur la figure 5.1 qui rendait difficile une adaptation fine de l'application à la topologie de la machine.

Près de 10 ans après, les processeurs généralistes Intel et AMD utilisent les mêmes concepts. Les contraintes se sont légèrement assouplies, permettant des topologies mémoire faciles à modéliser dans les cas les plus courants. [1] Malheureusement, les BIOS et les systèmes d'exploitation exposent rarement des informations complètes de topologie NUMA et les logiciels comme hwloc ne peuvent pas toujours en bénéficier pour préciser leur modèle. [2]

1. Les processeurs Intel et AMD disposent de 4 liens d'interconnexion, ce qui permet de créer des graphes complets jusqu'à 4 processeurs. Au-delà, on revient souvent aux technologies historiques basées sur un réseau externe (comme SGI NumaLink dans les plates-formes Altix UV) dont la topologie est un anneau ou un hypercube.

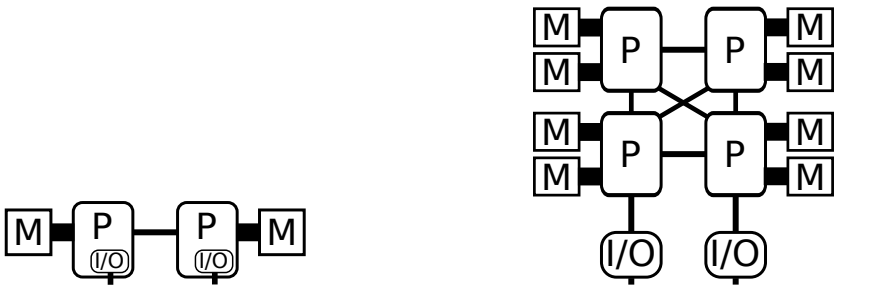
2. Les matrices de distance NUMA sont souvent simplistes et les informations de routage mémoire difficilement accessibles.

5.1.2 Intégration des puces... et des contraintes de localité

La démocratisation des architectures NUMA dans les serveurs généralistes fait en fait partie d'une longue course à l'intégration des composants annexes dans les microprocesseurs. En effet, il y a une vingtaine d'années, le coprocesseur arithmétique (l'unité de calcul flottant) est devenu standard en entrant dans le processeur principal. Quelques années plus tard, ce sont les caches (désormais plusieurs niveaux) qui ont subi le même sort. Une raison à cette évolution est que la miniaturisation permet d'intégrer plus de transistors dans les processeurs. Et cette intégration réduit les distances physiques et améliore donc les performances. Par ailleurs, elle permet aux constructeurs de mieux contrôler l'environnement entourant leurs processeurs et réduire les contraintes de compatibilité.

Avec l'avènement du multicœur et l'impossible continuation du modèle du contrôleur mémoire centralisé, il a semblé naturel d'en intégrer un dans chaque processeur. Les cœurs accédant essentiellement leur mémoire locale, le réseau entre les processeurs est censé être peu utilisé... à condition que les applications sachent bien utiliser ces architectures en plaçant les données près des tâches qui y accèdent.

Une nouvelle étape vient d'avoir lieu dans cette évolution avec l'intégration du contrôleur d'entrées-sorties (*PCI Express*) dans les processeurs Intel *Sandy-Bridge*. [3] Là aussi, la nécessité de rapprocher les cœurs des périphériques pour améliorer les performances des entrées-sorties amène un nouveau problème de localité. Ces architectures que nous avons nommées NUIOA (*Non Uniform Input/Output Access*) existaient déjà préalablement (figure 5.1), mais se généralisent aujourd'hui aux plates-formes de calcul les plus courantes comme en témoigne la figure 5.2. Nous avons ouvert un domaine de recherche autour de l'adaptation des communications MPI à ces nouvelles contraintes avec ma doctorante Stéphanie Moreaud, comme nous le verrons dans la section 5.3.



(a) Serveur à 2 processeurs Intel Xeon E5-2600 (2012).

(b) Serveur à 4 processeurs AMD Opteron 6200 (2012).

FIGURE 5.2 – Topologie NUMA et NUIOA des plates-formes les plus courantes en HPC en 2012.

Quelle est la prochaine étape ? Après le rachat par Intel de la division InfiniBand de QLogic et de la division réseau de Cray, on s'attend à ce que Intel intègre maintenant les interfaces réseau dans le processeur pour réduire encore plus le chemin critique entre le processeur et les périphériques. Une autre possibilité consisterait à supprimer les interfaces réseau et à étendre le bus d'entrées-sorties *PCI express* à des communications entre serveurs. Dans les deux cas, le réseau serait connecté à un processeur de manière très privilégiée, et donc les tâches communicantes devront faire très attention à s'exécuter sur ce processeur. Intel a confirmé son intention de suivre une telle direction avec l'annonce fin 2013 de l'intégration du réseau dans la prochaine génération de processeur Xeon Phi *Knights Landing* en 2015.

3. AMD devrait suivre cette voie en 2015.

5.2 Migration mémoire

Devant la complexité des architectures NUMA et les contraintes qu'elles imposent sur la localité des données, je me suis tout d'abord intéressé au placement et à la migration des données sur ces plates-formes.

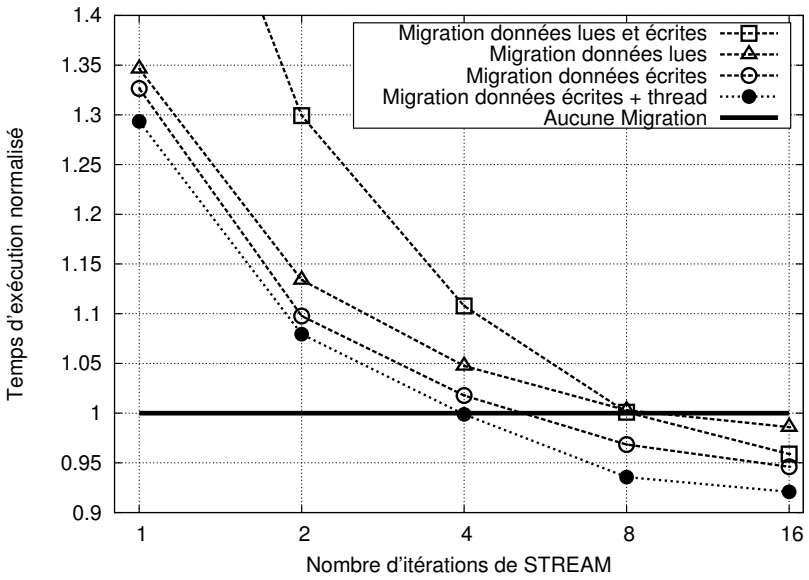


FIGURE 5.3 – Influence de la migration sur les performances du benchmark STREAM [52] sur une machine à 4 processeurs quadricœurs AMD Opteron 8347HE. Un thread manipule deux buffers en lecture (un local et un distant) et un en écriture (distant). On compare les performances selon si le thread accède directement aux données distantes ou si le thread et/ou certains buffers sont migrés pour améliorer la localité. Le coût des migrations est amorti quand le nombre d'itérations de calcul augmente, il est prohibitif initialement mais devient ensuite intéressant.

Prendre des décisions de placement mémoire peut sembler similaire (et lié) au placement de tâches. Le problème est en fait plus complexe. D'une part, les coûts de migration de données sont beaucoup plus importants et dépendent de la taille. D'autre part, les accès aux données n'ont pas les mêmes contraintes en lecture et en écriture. La figure 5.3 illustre la complexité des décisions dans un test mettant en jeu un thread et trois zones de données. Par ailleurs, les politiques de placement mémoire sont plus variées puisqu'il est possible de répartir les données sur plusieurs bancs mémoire (pour maximiser la bande passante disponible), de les placer dynamiquement lors du prochain accès, voire même de les répliquer à différents endroits (données en lecture seule).

Si les systèmes d'exploitation et les logiciels comme hwloc savent appliquer certaines politiques de placement ou migration mémoire, nous avons dû d'une part étendre et optimiser ces fonctionnalités, et d'autre part proposer des critères de décision. Ces travaux ont été menés dans le cadre de la thèse de François Broquedis dont l'objet était l'ordonnancement de threads OpenMP sur plates-formes hiérarchiques [14]. [4]

4. Thèse encadrée par Raymond Namyst et Pierre-André Wacrenier après un stage de master recherche encadré par Raymond et moi-même.

5.2.1 Localité des données et dynamicité

Le placement des données se fait généralement en suivant le placement des tâches afin de garder les données proches de leurs utilisateurs. Lorsque le placement des tâches est statique, celui des données peut également l'être, et peut donc être mis en place à l'initialisation du programme. Cependant, dans de nombreux cas, en particulier dans les applications multithreadées, les tâches travaillant sur chaque bloc de données ne sont pas toujours les mêmes. Un exemple classique concerne l'initialisation : les données sont lues depuis le disque par un seul thread avant d'être manipulées par plusieurs threads d'une section parallèle.

En effet, les différentes phases des applications ne seront pas toujours parallélisées de la même façon car leur importance et leur complexité varie. Pourquoi s'embêter à paralléliser la lecture depuis le disque si le goulet d'étranglement est le disque ? Pourquoi passer du temps à paralléliser une phase très courte et très complexe si l'impact est négligeable sur la performance globale ? Le nombre de tâches impliquées peut donc varier au cours de l'exécution, et les données vont donc avoir des utilisateurs (et donc des contraintes de localité) variables.

On sait depuis longtemps que la localité des données est un critère important pour l'ordonnancement [62]. Il faut également y veiller tout au long de l'exécution. Inutile de préciser que le placement statique initial peut être difficile à conserver si l'application est irrégulière. Il va falloir décider entre migrer des tâches près des données et le contraire. Prendre de telles décisions impose d'identifier les données utilisées par les différentes tâches et de quantifier les accès afin de fournir des critères de choix.

L'environnement de programmation va avoir des conséquences importances sur ce problème. Dans une application programmée entièrement à la main, de la création des threads jusqu'au placement, le programmeur maîtrise suffisamment la plate-forme pour optimiser la localité. Il connaît son application, le schéma de communication, et peut donc tenter d'en déduire quelles tâches ou données migrer lors des changements de phase. Par contre, quand l'application utilise des bibliothèques parallèles, en particulier un runtime OpenMP, le problème devient très difficile car la connaissance de l'application n'est pas disponible dans le runtime qui prend les décisions de placement.

5.2.2 Gérer les affinités dans le compilateur ou dans le runtime ?

La première solution consiste à exposer à l'application des fonctionnalités de placement afin qu'elle aide le runtime. Cette approche plutôt statique car essentiellement gérée à la compilation, semble être l'axe privilégié par les évolutions récentes ou prochaines du standard OpenMP [23]. Malheureusement, jusqu'à présent, les politiques offertes sont très simples et pas toujours portables (voire spécifiques aux implémentations). Effet, elles imposent au développeur de préciser manuellement la taille des sections parallèles, ou des groupes de threads, ou les numéros de cœurs sur lesquels placer les threads. Ces informations doivent être adaptées à toute nouvelle plate-forme selon sa taille ou sa numérotation des ressources. Les politiques de plus haut niveau sont actuellement en discussion [5] avec pour objectif de permettre au développeur de spécifier qu'il souhaite optimiser le partage de données (*compact* ou *close*) ou à maximiser la bande passante mémoire (*scatter* ou *spread*).

La solution alternative que l'équipe Runtime a promu, avant même que je la rejoigne, consiste à faire descendre des informations d'affinité de l'application vers le runtime pour l'aider à prendre des décisions dynamiquement [6]. Après la thèse de Samuel Thibault qui s'in-

5. Mais elles ne seront pas intégrées à la prochaine révision 4.0 du standard.

6. Cette approche ne semble pas populaire auprès de l'OpenMP ARB (*Architecture Review Board*) qui maintient le standard. Une des raisons officieuses est que les membres de l'ARB viennent du domaine de la compilation et préfèrent donc des approches statiques plutôt que d'imposer un runtime complexe gérant des approches dyna-

téressait à l'ordonnancement de threads sur plates-formes hiérarchiques [68], nous avons appliqué ces idées à l'ordonnancement conjoint des threads OpenMP et de la mémoire dans le cadre de la thèse de François Broquedis [14]. Il s'agit donc d'ajouter des directives OpenMP pour indiquer quelles zones mémoire sont utilisées et comment [WI1, RI1]. Cela permet donc au runtime de distribuer et/ou migrer les threads et les données en respectant dynamiquement leurs affinités et la localité des données, sans que ces directives ne dépendent de la plate-forme matérielle utilisée. Pour ce faire, des ordonnanceurs de threads ciblant spécifiquement les affinités de cache et de mémoire pour plate-forme NUMA ont été conçus [CI2].

Externe × Interne	GCC `libgomp3`	Intel ICC	ForestGOMP
4×4	9,4	13,8	14,1
16×1	14,1	13,9	14,1
16×2	11,8	9,2	14,1
16×4	11,6	6,1	14,1
16×8	11,5	4,0	14,4
32×1	12,6	10,3	13,5
32×2	11,6	5,9	14,2
32×4	11,2	3,4	14,3
32×8	10,9	2,8	14,5

TABLE 5.1 – Speed-up du benchmark NAS BT-MZ (classe C) avec ForestGOMP et les supports OpenMP de GCC et ICC, par rapport au temps d'exécution séquentiel, sur une machine à 4 nœuds NUMA quadricœurs, en fonction du nombre de threads dans les boucles parallèles OpenMP externes et internes.

Ces travaux ont été implémentés dans le runtime ForestGOMP que nous avons créé en adaptant le support OpenMP du compilateur GCC au dessus de notre bibliothèque de threads hiérarchique Marcel. La thèse de François a obtenu des résultats significatifs sur différentes applications irrégulières grâce à une conservation des informations d'affinité tout au long de l'exécution. Là où les implémentations OpenMP classiques utilisent des solutions basiques visant soit à placer les threads à leur initialisation, soit à les répartir la charge sans tenir compte des affinités, ForestGOMP est capable de réajuster dynamiquement la répartition de charge tout en conservant la localité des threads proches et celle de leurs données. La table 5.1 montre les résultats sur l'application BT-MZ (NAS multizone) dont le parallélisme externe est irrégulier et implique de répartir la charge entre les équipes de threads de la boucle parallèle interne. ForestGOMP parvient à conserver une très bonne accélération (14-15 sur 16 cœurs) quand le nombre de threads augmente, contrairement à GCC et ICC dont les politiques de placement et répartition sont trop simples ou statiques.

Avec des informations de performance mémoire de la plate-forme, ForestGOMP peut prendre des décisions de migration complexes comme illustré sur la figure 5.3. Ces travaux ont par ailleurs l'avantage d'être portables puisque l'application donne des informations d'affinités sans rien imposer sur le nombre de threads OpenMP ou leur placement. Ils s'appliquent même aux applications dont l'irrégularité est dynamique car dépendante des données en entrée, comme par exemple l'*Adaptive Mesh Refinement* qui raffine les calculs autour des singularités.

miques.

5.2.3 Améliorer le support système pour de placement mémoire

Les travaux de thèse de François Broquedis ont souvent soulevé des besoins au-delà de l'interaction entre applications et runtime. En effet, le runtime OpenMP s'appuie sur le système d'exploitation pour placer et migrer les tâches et données et nous nous sommes heurtés à des limites techniques. En effet, s'il est important de connaître la quantité d'accès mémoire et leur vitesse pour prendre des décisions, encore faut-il que la migration soit efficace.

Nos premières expérimentations semblaient rendre la migration mémoire inutile car beaucoup trop chère, sauf pour de très grands volumes de données (plusieurs dizaines de megaoctets). Ce résultat n'est pas gênant en soi, il aurait simplement indiqué que nos ordonnanceurs devraient placer les tâches selon les données, et pas le contraire. Cependant nous avons montré que cette lenteur était en fait un défaut d'implémentation (nos correctifs sont totalement intégrés à partir du noyau Linux 2.6.31).

L'autre axe sur lequel nous avons dû travailler est la disponibilité de fonctions de migration avancée. En particulier nous avons implémenté un mécanisme de *migration-on-next-touch* qui permet aux pages mémoire de suivre les migrations du prochain thread qui y accède. Cette fonctionnalité permet de redistribuer facilement une grande zone mémoire entre différents threads sans avoir à préciser à l'avance qui va accéder à quelle partie, tout est fait dynamiquement au dernier moment. Si l'idée n'est pas neuve, parfois appelée *Lazy Migration*, c'est un marronnier qui revient régulièrement dans les articles de recherche sans jamais avoir été implémenté à grande échelle [7]. Nous l'avons implémentée de manière élégante [8] et présentée à la communauté Linux [CI9]. Malheureusement, cette fonctionnalité étant alors spécifique aux applications de calcul haute performance, elle a été considérée comme trop peu utilisée pour être intégrée dans le noyau officiel.

Quelques années plus tard, l'horizon s'est éclairci car les architectures NUMA se sont généralisées et des applications de virtualisation commencent à avoir des besoins similaires. Deux propositions concurrentes de mécanismes permettant aux données et tâches de se suivre automatiquement ont été soumises récemment [9] et une première avancée est attendue dans le noyau 3.13 début 2014. Comme souvent, le HPC pourrait bénéficier de mécanismes que d'autres domaines auront fini par imposer à sa place, on y reviendra dans la conclusion 6.3.2.

5.2.4 Bilan sur la migration mémoire

Nos activités de recherche sur la migration mémoire ont obtenus des résultats significatifs, comme en témoignent toutes les publications [WI11, WI4, WI1, CI9, CI2, RI1] mais nous avons quasiment échoué à les imposer à la communauté.

Tout d'abord, nos propositions d'extension du standard OpenMP pour gérer les affinités mémoire sont restées lettre morte car le standard est maintenu par une communauté proche de la compilation qui n'est pas prête à intégrer des stratégies dynamiques complexes dans ses runtimes. Le standard semble considérer plus important aujourd'hui de gérer les architectures hétérogènes comme les GPGPUs que de gérer correctement la localité des données. Cependant, comme le modèle OpenMP est très loin du modèle de programmation des GPGPUs, cela aurait imposé des modifications très profondes d'OpenMP. Les extensions pour accélérateurs introduites dans OpenMP 4.0 ne ciblent finalement que le Xeon Phi d'Intel, accélérateur beaucoup plus proche d'un processeur généraliste multicœur que d'une carte graphique.

7. Seul le système d'exploitation Solaris le propose en standard depuis longtemps.
8. Nous avons imité le mécanisme classique du *Copy-on-write* qui duplique les pages au dernier moment lors de la création d'un processus Unix avec `fork`.
9. `http://lwn.net/Articles/486858/` propose le concept du *Home-node*, tandis que `http://lwn.net/Articles/488709/` présente *AutoNUMA*.

Il me paraît difficile pour OpenMP de rattraper l'avance prise par les standards concurrents pour les GPGPUs. OpenCL et OpenACC restent les deux seuls "standards" de programmation des GPGPUs, ce dernier ayant été créé en attendant un hypothétique intégration dans OpenMP. OpenMP risque donc non seulement d'échouer dans cette nouvelle cible, et en plus d'échouer à gérer correctement son cœur de cible, les machines à mémoire partagée où la localité des données est de plus en plus critique. OpenMP continuerait donc à être la solution de parallélisation du pauvre : ça marche à peu près, mais ça n'est jamais parfait, et en particulier sur ces plates-formes NUMA ou sur les accélérateurs et GPUs.

Nos améliorations du support pour la migration mémoire dans le noyau Linux n'ont que partiellement été intégrées au noyau officiel. Cependant, il y a là de l'espoir car d'autres communautés ont désormais des besoins similaires. Une solution proche de la migration fainéante a finalement été intégrée dans Linux 3.13 après plus d'un an de discussion entre plusieurs développeurs noyau reconnus, preuve que c'est un problème difficile. [10]

Nos travaux ont par ailleurs mis en exergue la complexité de concevoir une interface de placement mémoire comme nous le supposions en introduction de 5.2. En effet, non seulement les possibilités de placement souhaitées sont vastes, mais en plus les systèmes d'exploitation en proposent des variantes très différentes. Par exemple, l'interface générique de placement mémoire de hwloc a demandé un gros travaux d'uniformisation de toutes ces idées. Alors que le placement de tâche ne propose que 2^3 variantes [11], le placement mémoire propose 6 politiques avec 2^4 variantes pouvant être appliquées à différentes zones mémoire. La complexité de cette interface n'est pas satisfaisante pour l'utilisateur lambda [12] mais est nécessaire pour couvrir tous les besoins. Cela ne fait que confirmer que prendre des décisions de placement mémoire est un problème difficile.

Une fois ces fonctionnalités de migration avancées disponibles, il restera le problème de décider entre les différentes possibilités de migration des tâches et/ou des données. Là encore, cela impose la disponibilité d'informations quantitatives comme discuté en section 4.4.1.

5.3 Adaptation des communications aux contraintes NUMA

Si les architectures NUMA existent depuis une vingtaine d'années, elles sont réellement démocratisées pendant mon post-doctorat puis mon début de carrière à Inria avec leur apparition dans les processeurs généralistes d'AMD et Intel. Leur présence dans les grappes de calcul a ensuite conduit à la généralisation des affinités des périphériques, ce que nous avons appelé NUIOA, pour *Non Uniform Input/Output Access*, en section 5.1.2. Dans le cadre de la thèse de Stéphanie Moreaud [53], nous avons été les pionniers à étudier ce nouveau problème. Nous avons notamment montré son importance pour les communications MPI, mais nos travaux dans KNEM (chapitre 3) l'ont également mis en évidence pour les moteurs DMA d'I/O AT, [13] tandis que nos collègues travaillant sur StarPU l'ont rencontré pour les GPUs.

5.3.1 Différents angles d'approche

L'influence de la localité sur les entrées-sorties peut se décliner sous deux aspects. D'une part, on peut utiliser les informations de localité lors du placement des processus. Un exemple simple consiste ainsi à placer une tâche proche des périphériques qu'elle utilise. D'autre part,

10. Le support des architectures NUMA dans Linux en juillet 2013 est détaillé dans [45].
11. Placement de thread ou de processus, strict ou non, avec ou sans effet sur le placement mémoire.
12. hwloc fournit donc des routines de placement *basiques* pour masquer cette complexité.
13. J'ai d'ailleurs amélioré la façon dont le noyau Linux alloue les canaux DMA pour copie déportée afin de respecter les affinités. Ce changement est inclus dans la version 3.12.

une fois le placement des tâches effectués, on peut essayer d'en tenir compte dans la façon dont elles utilisent les périphériques. Il s'agit par exemple de choisir les stratégies de communications selon la distance des périphériques afin d'éviter des saturations sur les bus impliqués.

Le choix entre un axe et l'autre tient des autres contraintes imposées sur le système. Par exemple, si l'utilisateur impose des contraintes de placement indépendantes de la localité des périphériques, le second axe devra être privilégié. C'est par exemple le cas quand un algorithme de placement tel que TreeMatch (voir en section 4.1.2) n'observe que les affinités entre tâches et données mais pas celles pour les périphériques. La localité des périphériques ne pourra être prise en compte qu'après le placement de TreeMatch en essayant d'adapter les communications à ces contraintes.

Si par contre, le placement est initialement libre, les contraintes de localité des périphériques peuvent être prises en compte dès le début. Mais leur prise en compte ne garantit pas un placement optimal. Il peut par exemple exister des contraintes contradictoires quand un processus souhaite être proche de deux périphériques distants. Il faudra donc parfois faire des choix lors de l'application du premier axe, et dans tous les cas étudier l'intérêt du second axe pour compenser les lacunes du premier.

Toutes ces idées s'appliquent aux différents types de périphériques, mais en particulier ceux dont la performance est critique. En calcul haute performance, cela inclut notamment les cartes réseau ou InfiniBand et les GPGPUs. Je décris ci-après les contributions que nous avons apportées sur les communications réseau dans le cadre de la thèse de Stéphanie Moreaud.

5.3.2 Quel impact pour la localité des cartes réseau ?

À mon arrivée dans l'équipe Runtime en 2006, un des axes d'études était le multirail MPI, c'est-à-dire l'utilisation de plusieurs cartes réseau pour les communications dans les applications parallèles [8]. Si la nécessité de verrouiller les tâches sur un processeur pour obtenir des performances reproductibles commençait à être connue par la communauté, elle ne suffisait pas à expliquer les variations de performances observées par mes collègues. Les performances étaient reproductibles pour un cœur donné, mais le débit réseau observé au niveau de l'application variait très significativement d'un cœur à un autre, jusqu'à 40% comme montré sur la figure 5.4. Ce résultat a confirmé la nécessité d'étudier l'impact de la localité des périphériques que j'avais mis en avant lors de mon recrutement à Inria.

La localité des périphériques, c'est-à-dire le fait qu'ils soient potentiellement plus proches de certains cœurs que d'autres, a commencé à se généraliser dans le HPC en 2004 avec le succès des processeurs AMD Opteron. Elle s'est aujourd'hui définitivement installée comme expliqué en section 5.1.2. Nous avons cherché dès 2006 à quantifier l'impact de ce *Non Uniform Input/Output Access* (NUIOA) sur les communications réseau et à y proposer des solutions. Nous avons mis en évidence un certain nombre de phénomènes concernant la latence et sur le débit des communications dépendants de la distance entre le périphérique réseau et la mémoire contenant les données transmises [CI13], et qui dépendent de plus de la plate-forme comme nous l'avons constaté plus tard.

En pratique, les effets sont difficiles à quantifier et à modéliser. Ils sont liés à des caractéristiques matérielles telles que le nombre de requêtes matérielles pouvant être simultanément en cours sur les bus mémoire ou d'entrées-sorties. Ils peuvent de plus dépendre du type de communication utilisé par le périphérique (PIO ou DMA, par paquets de quelle taille...) et de la direction (entrée ou sortie). Ceci nous conduit à une situation où les effets existent principalement pour les périphériques haute performance comme les GPUs et les cartes InfiniBand mais peuvent difficilement être quantifiés.

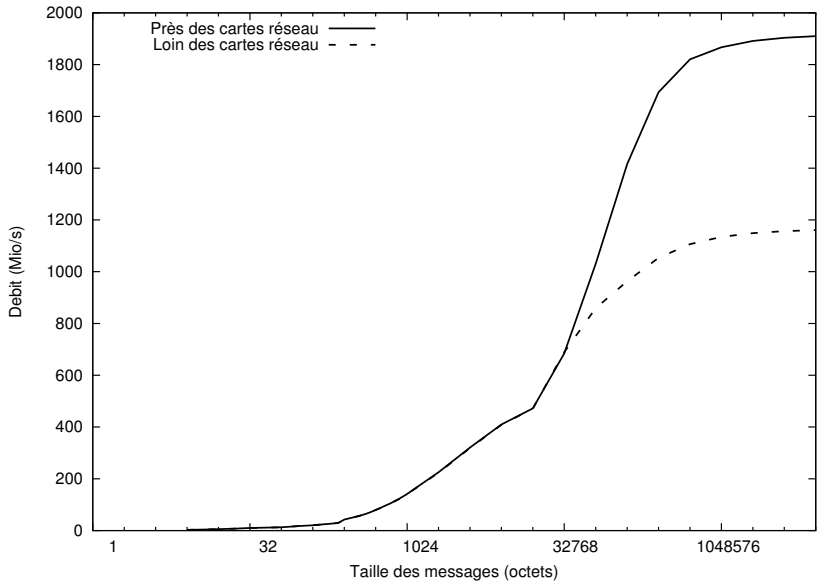


FIGURE 5.4 – Débit d'un ping-pong MPI entre deux machines à deux processeurs Opteron bicœurs reliées par une carte QsNet II et une carte Myri-10G, selon si les processus s'exécutent sur des cœurs proches ou loin de ces cartes.

5.3.3 Placer les tâches communicantes selon la localité des périphériques

Si les performances de communications MPI souffrent de la distance entre la mémoire et les périphériques, il est important de faire en sorte de réduire cette distance. Nous avons mis en place des techniques permettant aux bibliothèques de communication et aux applications de connaître la localité des périphériques afin de se verrouiller sur un cœur proche. Avant de disposer aisément de telles informations dans des outils comme hwloc [14], il a fallu ajouter un certain nombre de mécanismes aux couches logicielles.

Tout d'abord, il a fallu modifier le système d'exploitation pour exposer les informations de localité des périphériques matériels. [15] Ensuite, il faut être capable de faire correspondre ce périphérique matériel avec ce que l'application utilise : quand on ouvre un socket, on ne sait pas directement quelle interface réseau il va utiliser, et on ne sait pas à quel périphérique PCI ces interfaces correspondent ! Si certains pilotes standards comme InfiniBand fournissent cette correspondance, d'autres comme le pilote MX des cartes Myri-10G ont nécessité que nous l'ajoutions.

Une fois les informations de localité disponibles, il est facile de placer les tâches proches des périphériques qu'elles utilisent. La question qui se pose alors est de savoir qui doit prendre en charge ce placement. Pour les micro-benchmarks dont l'objectif est d'obtenir une performance reproductible et optimale, le placement est fait manuellement par l'utilisateur. Pour une application MPI qui ignore tout du réseau matériel sous-jacent, c'est plus difficile. La solution consiste à manipuler les informations de placement là où les périphériques sont réellement utilisés, notamment dans la bibliothèque MPI. C'est ce que nous avons implémenté dans la bibliothèque de communication NewMad de l'équipe [CI13]. Cette solution à l'avantage de s'intégrer aisément dans les stratégies habituelles de placement qui sont également mises en

14. hwloc dispose de fonctions retournant directement l'ensemble des cœurs proches d'un périphérique logiciel tel que l'interface réseau `eth1`, la carte InfiniBand `mlx4_2` ou le 3ème *device* CUDA.

15. J'ai ajouté un attribut `sysfs` dans le noyau Linux officiel 2.6.22 pour indiquer le nœud NUMA proche de chaque périphérique PCI.

place dans les implémentations MPI.

Si le placement des tâches près des cartes réseau permet d'améliorer les performances des communications, il ne doit pas nuire à l'équilibrage de charge. Les serveurs de HPC modernes disposent en général de deux processeurs et d'une carte InfiniBand proche de l'un d'entre eux. Seule la moitié des cœurs pourra donc avoir un accès privilégié au réseau, il faut donc être capable d'identifier les processus MPI utilisant le réseau plus intensivement le réseau pour ensuite les privilégier. Une solution consiste à étendre les algorithmes de placement tels que TreeMatch. Une fois les affinités entre tâches prises en compte pour les regrouper par affinités et privilégier les communications intra-nœud, on peut observer les communications inter-nœud résiduelles pour définir l'affinité des tâches pour le réseau. Privilégier l'accès réseau de certains processus revient à *retourner* certaines branches de l'arbre de placement (comme représenté sur la figure 5.5), ce qui ne change rien à la gestion des affinités intra-nœud de TreeMatch, mais permet de modifier la distance entre certaines branches et certains périphériques. C'est une approche que nous commençons à étudier.

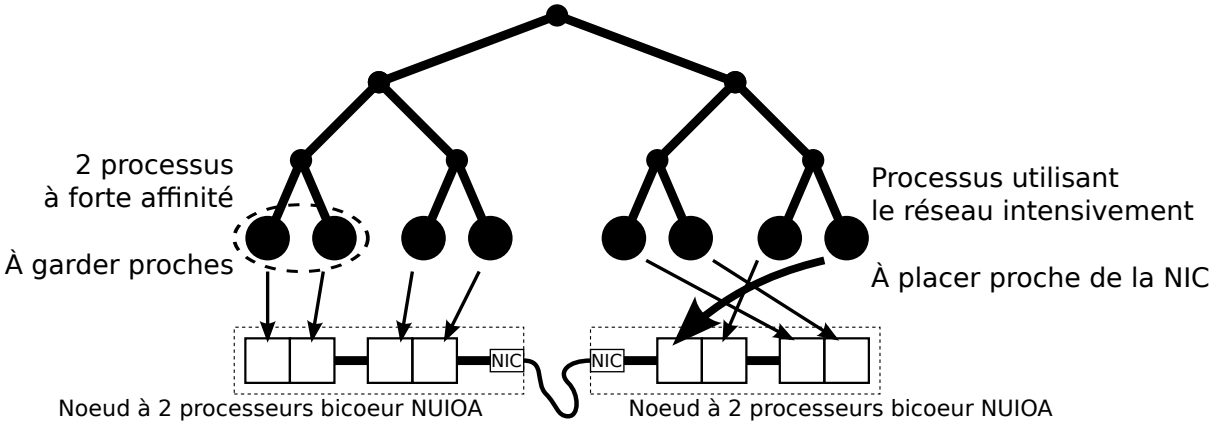


FIGURE 5.5 – Placement de processus MPI groupés par affinité en ajoutant des contraintes NUIOA. On retourne la moitié droite de l'arbre d'affinités, ce qui ne modifie pas sa structure, mais permet de donner un accès privilégié au réseau à un processus l'utilisant intensivement.

Dans le cas d'applications hybrides MPI+OpenMP, ce sont les threads communicants qui devront être privilégiés. Dans le cas où un seul thread maître se charge des communications MPI entre les sections parallèles OpenMP, l'implémentation MPI en sera informée (via l'initialisation avec `MPI_THREAD_FUNNELED`). Elle pourra alors le placer près du réseau, à condition de ne pas contredire les décisions de placement du runtime OpenMP. Mais une telle collaboration est actuellement une utopie car aucune implémentation intégrant MPI et OpenMP suffisamment finement n'est disponible.

5.3.4 Adapter les communications au placement

Les affinités des tâches pour les cartes réseau peuvent être difficiles à mesurer mais aussi être inexploitables car contradictoires. Par exemple si deux cartes InfiniBand sont disponibles mais distantes l'une de l'autre, un processus ne pourra pas avoir un accès privilégié aux deux simultanément. Par ailleurs, les affinités entre tâches et périphériques peuvent varier au cours de l'exécution, par exemple si l'application est constituée de phases très différentes. Une solution consiste alors à modifier dynamiquement le placement des tâches selon l'évolution des affinités. Dans le cas de MPI, cela peut également consister à renuméroter les processus pour confier les rôles communicants fortement aux processus près du réseau. Cependant, ces opé-

rations ont un coût non négligeable et deviennent inenvisageables quand les changements d'affinités sont trop fréquents. Il convient alors d'étudier notre second axe, l'adaptation des stratégies de communication à un placement donné.

Notre première proposition a consisté à étudier l'utilisation de cartes réseau multiples dans un contexte NUIOA. Les implémentations MPI découpent habituellement les messages de manière équitable afin de répartir le débit réseau sur les différents *rails* disponibles. Nous avons montré que cette stratégie n'est pas optimale et qu'il faut en fait privilégier partiellement la carte la plus proche du processus considéré. Cela permet un gain de 15% sur des micro-benchmarks simples. Ensuite nous avons précisé ce résultat dans le cadre d'un schéma de communication intense en montrant que l'affinité NUIOA ne doit pas être considérée dans tous les cas [CI14]. La figure 5.6 montre que sur une machine à quatre processeurs, quand tous les cœurs communiquent en même temps, les cœurs loin des cartes doivent utiliser équitablement les deux cartes (le maximum selon l'axe gauche se trouve à 50%), tandis que les cœurs proches d'une carte devront utiliser uniquement celle-ci (le maximum selon l'axe droit se trouve à 100%).

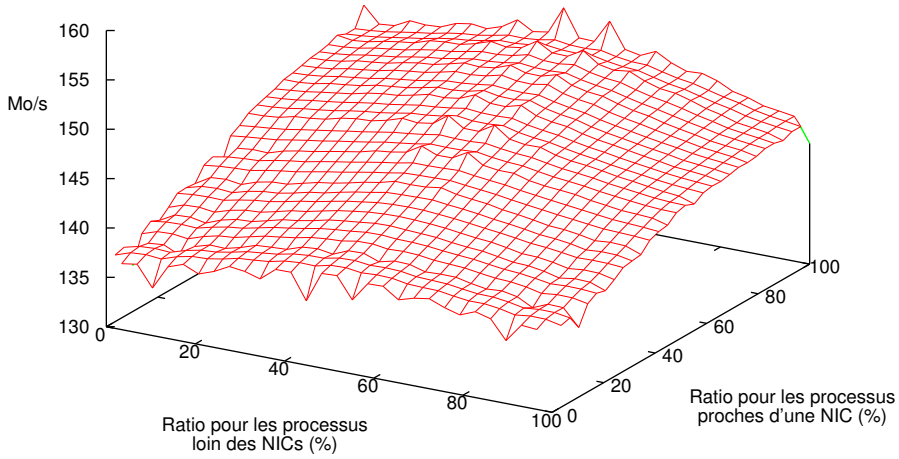


FIGURE 5.6 – Débit (par processus) lors d'une opération collective MPI *Alltoall* entre 16 processus répartis sur 2 nœuds disposant de 2 cartes InfiniBand, selon les ratios de répartition des messages entre les deux cartes pour les processus loin ou proches d'elles.

Nous avons également montré que les contraintes d'affinités réseau pouvaient être prises en compte dans les opérations MPI hiérarchiques. Une implémentation courante de ces opérations consiste à d'abord agréger les contributions locales à l'intérieur de chaque nœud avant de les échanger sur le réseau. Cela implique l'élection d'un processus leader intermédiaire sur chaque nœud. Nous avons montré que les stratégies d'élection habituelles (qui cherchent à répartir la charge) doivent être revues pour forcer l'élection d'un leader proche du réseau car il communique beaucoup plus (voir la figure 5.7). Cette idée permet d'améliorer sensiblement les opérations collectives *rootées* (ayant une racine, par exemple Broadcast, Scatter ou Gather) [WI5].

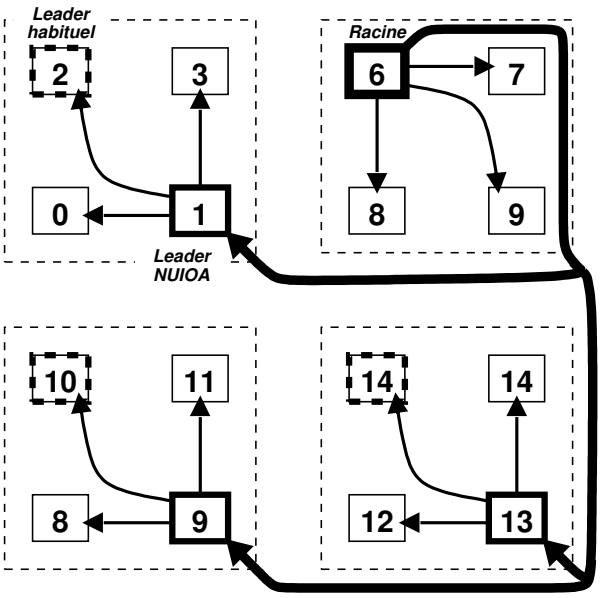


FIGURE 5.7 – Implémentation d'une collective MPI *Broadcast* hiérarchique avec élection NUIOA des leaders intermédiaires sur chaque nœud. Au lieu d'utiliser sur chaque nœud un leader dont le rang local est le même que celui de la racine du Broadcast (en haut à gauche), on force l'élection d'un leader près de la carte réseau (en bas à droite).

5.3.5 Bilan sur les affinités des périphériques

Je suis fier de nos travaux sur l'affinité des périphériques pour deux raisons. D'une part, nous avons été les pionniers de cette étude (et avons inventé le sigle NUIOA). D'autre part, nous avons standardisé les outils permettant aux développeurs d'en tenir compte. Le noyau Linux standard et un logiciel comme hwloc permettent désormais d'obtenir très facilement la localité des interfaces des réseaux rapides du HPC. Les implémentations MPI reposant quasiment toutes désormais sur hwloc, elles peuvent utiliser ces informations de localité pour implémenter les idées que nous avons proposées, et Open MPI le fait d'ores et déjà. [16]

Le cas des autres périphériques, en particulier les GPGPUs, est en passe d'être également résolu puisque nos contacts avec NVIDIA dans le cadre de la thèse de Cédric Augonnet nous ont permis de leur faire ajouter les fonctionnalités nécessaires à l'interface CUDA (et AMD propose depuis peu une extension OpenCL similaire). Il reste maintenant à mettre en place des stratégies adaptées à ces périphériques. Sur de nombreux serveurs modernes, cela pourra amener à choisir entre privilégier l'affinité pour un GPGPU et celle pour une carte InfiniBand. On s'attend à ce que l'affinité du GPGPU soit plus critique sur les performances globales de l'application car il est souvent utilisé de manière étroitement liée au CPU local, mais ce problème méritera d'être étudié plus finement.

Par contre, la déception dans notre travail est que nous n'avons pas suffisamment étudié l'impact de nos idées sur des applications réelles. Il reste également à creuser l'intégration des contraintes NUIOA dans les algorithmes de placement existants comme TreeMatch. Sur les applications que TreeMatch peut optimiser significativement, nos travaux risquent d'avoir un impact limité car TreeMatch aura fortement réduit la quantité de communication inter-nœud.

Si les gains apportés par nos idées existent, ils restent souvent faibles tant que l'application n'utilise pas de très gros messages MPI. Cependant nos publications sur le sujet [CI13, CI14,

16. http://blogs.cisco.com/performance/process-affinity-hop-on-the-bus-gus/

WI5] ont, je crois, fait prendre conscience à la communauté de l'importance de prendre en compte ces idées. Après tout, la quête de l'*Exascale* s'annonce si difficile que de telles petites optimisations ne seront pas négligeables.

5.4 Bilan et perspectives

L'intégration des différents composants dans les puces, en particulier les contrôleurs mémoire et d'entrées-sorties, accroît l'importance de la localité dans les applications parallèles en rendant les plates-formes NUMA et NUIOA. J'ai proposé et implémenté différentes fonctionnalités permettant de simplifier la gestion de ces nouvelles contraintes. Cela nous a permis, d'une part, de faciliter le placement et la migration conjointe des tâches des données, et d'autre part d'adapter le placement et les communications réseau. Il reste maintenant du travail à faire pour intégrer ces contributions aux runtimes OpenMP et MPI. Si la diffusion d'hwloc rend nos résultats facilement réutilisables dans les runtimes, il faut faire remonter nos idées jusqu'à l'application, en particulier en améliorant la gestion des affinités dans les standards, mais la communauté n'y est pas encore toujours prête, comme expliqué à propos d'OpenMP en section 5.2.2.

Par ailleurs, certaines de nos fonctionnalités ne sont pas toujours largement disponibles. D'une part, il est difficile d'ajouter de nouvelles fonctionnalités au noyau Linux car le design de nouvelles interfaces de programmation est un processus complexe et critique. L'ajout d'un *flag* pour la migration paresseuse est par exemple encore en discussion 3 ans après nos travaux (mais l'avenir s'est heureusement récemment éclairci). Ajouter de nouveaux appels-système prend du temps car il faut s'assurer que l'interface est à la fois fiable et suffisamment évolutive. Un des obstacles à nos idées réside dans la conviction de certains développeurs Linux influents qu'on peut tout faire automatiquement dans le noyau, sans dialoguer avec l'application. On y reviendra en conclusion 6.3.2. Nous pensons que le noyau devrait au contraire fournir des interfaces étendues permettant d'exposer plus d'information aux utilisateurs sur l'état de la machine, et permettant aux applications d'agir plus directement sur le processus de décision. Les applications ou les bibliothèques pourraient ainsi contribuer à certaines décisions, car elles connaissent leurs propres besoins, contrairement au noyau qui ne peut qu'essayer de les deviner en appliquant des heuristiques.

D'autre part, les concepteurs de cartes mères sont très lents à implémenter les mécanismes permettant aux systèmes d'exploitation et aux logiciels comme hwloc de connaître les informations de localité NUMA et NUIOA utilisées dans ce chapitre. Beaucoup de machines ne respectent par exemple pas à la lettre les spécifications ACPI qui demandent d'exposer précisément les distances entre processeurs, mémoire et bus d'entrées-sorties. La situation s'améliore doucement depuis quelques années, notamment depuis que les plates-formes Intel sont devenues NUMA, mais il reste de nombreux vilains petits canards qui nous rendent la vie difficile et sur qui nous n'avons pas ou peu d'influence. La seule solution dans ce cas, pas du tout viable à long terme, consiste à contourner logiciellement ces problèmes en les listant exhaustivement.

Chapitre 6
Conclusion et perspectives

J'ai présenté dans ce manuscrit mes contributions aux supports exécutifs et systèmes d'exploitation pour le calcul haute performance. Elles se sont réparties en 4 chapitres visant d'une part à rendre plus accessibles et portables les techniques des réseaux haute performance (chapitres 2 et 3), et d'autre part à faciliter l'exploitation des architectures hiérarchiques complexes modernes (chapitres 4 et 5).

Avant de détailler quelques pistes pour le futur du HPC, je vais présenter un bilan de l'impact de mes travaux, aborder le sujet compliqué du développement logiciel et la communication autour de nos résultats, et les difficiles interactions entre notre communauté et celles qui gravitent autour.

6.1 Impact de mes travaux

L'impact des travaux scientifiques peut se juger au nombre de citations de nos publications. Mais dans des domaines aussi appliqués que le calcul haute performance, la diffusion des logiciels est souvent également un bon indicateur. La grande majorité de mes travaux s'est traduite sous la forme de nouveaux logiciels (ou briques logicielles) que j'ai veillé à rendre facilement utilisables via des bibliothèques très répandues, en particulier les implémentations MPI. Ces logiciels ont connu un succès assez significatif dans la communauté HPC, validant l'intérêt de mes travaux.

6.1.1 Une diffusion logicielle réussie

Le large et rapide succès d'hwloc (chapitre 4) a principalement été la conséquence de fonctionnalités longtemps espérées mais jamais implémentées, en particulier la portabilité vis-à-vis du système d'exploitation et de la numérotation des ressources. D'autre part, son intégration très rapide dans Open MPI et MPICH via nos collaborations existantes a été un tremplin pour sa diffusion. Cela a permis à de grandes communautés d'utilisateurs de connaître nos travaux rapidement, et d'en bénéficier facilement. On reviendra dans la section 6.2.2 sur le rôle qu'a joué la communication dans cette diffusion. La contrepartie à ce succès est qu'il faut maintenant assurer le support utilisateur, et ce n'est pas facile quand le logiciel n'est pas directement financé. En effet, hwloc est aujourd'hui un logiciel de support à la recherche plutôt que la cible des recherches elles-mêmes.

Dans le cas d'Open-MX (chapitre 2), la collaboration contractuelle avec Myricom (encore très populaire à cette époque) a apporté une caution à mes travaux de recherche permettant une diffusion plus rapide auprès d'une communauté qui aurait pu être sceptique à l'idée d'obtenir de bonnes performances sur Ethernet. Mais ce contrat a en revanche imposé d'insister sur la stabilité du logiciel tôt dans le cycle de développement, sans pour autant ralentir les activités de recherche. Les travaux sur Open-MX se reportent maintenant sur CCI dont l'objectif est de définir une interface générique pour différents domaines d'applications, pas seulement pour le HPC, et pour différents matériels réseau. Mais convaincre les utilisateurs sera ici un défi

plus difficile car ils devront convertir leurs codes avant d'utiliser CCI. Mais contrairement à Open-MX, les performances ne seront plus limitées par une interface trop proche des capacités matérielles des cartes Myrinet. L'avenir nous dira si CCI parviendra à convertir suffisamment d'utilisateurs pour convaincre la communauté de s'y intéresser plus largement.

Mes travaux sur les communications intra-nœud autour de KNEM (chapitre 3) ont servi de base à un certain nombre d'extensions par des équipes de recherche, notamment de l'Université du Tennessee (Knoxville). Le logiciel est facilement utilisable via MPICH ou Open MPI, il a suscité l'intérêt de Mellanox et Bull qui l'utilisent dans leurs distributions, mais sa popularité générale reste assez limitée. C'est notamment lié à la nécessité de faire installer le module noyau par un administrateur. Cependant, avec l'intégration récente de fonctionnalités similaires à KNEM dans le noyau officiel, on peut se satisfaire d'avoir pu populariser nos idées et fait naître le besoin de les fournir en standard sur n'importe quel cluster, ce qui était tout de même l'objectif initial : rendre accessible à tous une fonctionnalité initialement limitée aux pilotes des réseaux haute performance.

Sur les aspects NUMA et NUIOA, le bilan est plus mitigé. Concernant la gestion de la mémoire et des threads sur architecture NUMA (section 5.2), nous avons proposé des innovations pour répondre plus intelligemment aux difficultés, mais nous avons échoué à convaincre les communautés assez clivées du HPC et de Linux (on y reviendra en section 6.3). Concernant NUIOA (section 5.3), nous avons sensibilisé la communauté aux problèmes des affinités entre processeur, mémoire et périphériques, avons été les pionniers de leur étude, et avons mis en place les outils logiciels pour les résoudre. Il reste aux concepteurs d'algorithmes d'ordonnancement, placement, etc. à les prendre en compte, ce qui semble être de plus en plus souvent le cas, également grâce à hwloc.

6.1.2 Mais un impact limité aux spécialistes ?

Un reproche qu'on peut faire à mes travaux est qu'ils touchent peu les utilisateurs finaux et leurs applications. D'ailleurs, nous n'avons pas assez insisté sur l'impact de nos travaux sur les effets NUIOA au niveau des applications, et nous sommes cantonnés à un impact dans différentes petites couches. Mais plus généralement, nos travaux sont peu visibles des utilisateurs car ils se situent très bas dans la pile logicielle, comme représenté par la figure 1.1.

Les améliorations des communications réseau MPI avec Open-MX ou KNEM peuvent être constatées par les utilisateurs, mais encore faut-il qu'ils sachent comment les désactiver pour les comparer aux alternatives. [1] Quant aux apports de hwloc et NUIOA, ils sont globalement cachés dans des fonctionnalités internes des implémentations MPI, souvent assez obscures pour les utilisateurs, comme par exemple l'algorithme de placement de processus ou l'implémentation des opérations collectives. Un utilisateur avancé pourra étudier nos travaux à condition de ne pas être limité par la nécessité de disposer des droits administrateurs pour charger les modules noyau Open-MX et KNEM.

Ceci-dit, nos travaux apportent de petites pierres à l'édifice en optimisant certaines des nombreuses couches logicielles impliquées, et je ne pense pas qu'on puisse se passer de ce genre de travail dans la longue course vers l'Exascale. Il est vrai que mon travail se traduit le plus souvent en l'intégration de petites contributions dans des logiciels répandus (en particulier les implémentations MPI). C'est un point qui m'a beaucoup distingué des autres membres de l'équipe Runtime depuis mon arrivée à Inria en 2006. En effet, certains des axes de recherche que j'ai apportés pouvaient difficilement s'intégrer dans la suite logicielle PM2 qui regroupait alors presque tous les travaux de l'équipe.

1. Cela implique de recompiler MPICH ou de passer des options hermétiques à Open MPI, ce qui est plutôt réservé aux experts.

6.1.3 Améliorer les suites logicielles existantes ou développer la notre ?

Développer une grosse suite logicielle intégrée est un joli projet, qui peut montrer la cohérence d'une équipe de recherche. Mais cela demande énormément de ressources humaines. Quid du gâchis de temps d'ingénieurs si le succès n'est pas au rendez-vous ? Tout dépend de l'objectif.

Si l'on étudie un nouveau modèle de programmation, qu'aucun standard n'a clairement pris le pas, et qu'il y a peu de concurrents sérieux, développer son environnement de programmation parallèle du futur peut être une bonne idée. Même si le succès n'est pas au rendez-vous, on aura pu étudier notre idée de ce nouveau modèle.

Malheureusement, on a souvent tendance à vouloir redévelopper un tel logiciel sans véritable raison valable. Il y a une quinzaine d'années, les outils étaient moins matures et nombreux qu'aujourd'hui. Se faire une place était beaucoup moins difficile que dans la concurrence d'aujourd'hui. Cependant, il y a encore par exemple tous les ans plusieurs nouveaux logiciels d'ordonnancement de graphes de tâches qui apparaissent. Si les implémentations sont différentes, les idées et les algorithmes mis en jeu sont très similaires, car les fondements théoriques ont été établis il y a longtemps. Beaucoup de nos collègues auraient probablement mieux fait de contribuer à une implémentation existante (comme StarPU, développé dans l'équipe depuis bien plus longtemps) plutôt que de réinventer une roue quasiment identique dans leur coin. Malheureusement, nos egos (et d'éventuelles contraintes de propriété intellectuelle et de visibilité) nous donnent parfois de mauvais conseils.

L'avantage de développer une nouvelle suite est qu'on maîtrise l'intégralité du code (techniquement et en terme de propriété intellectuelle), qu'on est libre d'y faire ce que l'on veut. Mais les inconvénients sont trop souvent négligés :

— un gros temps de développement gaspillé à réimplémenter des choses déjà existantes, puis à essayer de les optimiser et stabiliser autant que les concurrents existants ;
— une maintenance future beaucoup plus lourde qu'il n'y paraît car les architectures et systèmes évoluent vite, et les utilisateurs sont exigeants, surtout si le succès est au rendez-vous ;
— un succès et une visibilité difficiles à atteindre car il faut se faire une place au soleil parmi les nombreux concurrents.

À l'opposé, intégrer nos travaux sous forme de petites briques dans des suites logicielles existantes coûte très peu en terme de développement et maintenance. Et cela assure une diffusion rapide si la suite est déjà reconnue. Cependant :

— notre travail pourra être rapidement oublié, dilué dans la popularité de la suite. Mais cela ne nous empêchera pas de rappeler son existence ;
— le temps passé à apprivoiser le code développé par d'autres n'est pas négligeable, mais il pourra rester faible si la suite est bien choisie et si on collabore explicitement avec ses développeurs.

Il y a aujourd'hui sur le marché deux implémentations MPI majeures (Open MPI et MPICH), la grande majorité des autres en est des dérivés (Intel MPI, Bullx MPI, MVAPICH, Cray MPI, Microsoft MPI, etc.). Il a donc été naturel pour moi d'y intégrer mes travaux plutôt que de redévelopper une autre implémentation, qui aurait été moins portable, moins stable, etc.

6.2 Au-delà de la recherche, des aspects logiciels, technologiques, communication et culturels

Il y a une vingtaine d'années, le HPC était essentiellement utilisé par des communautés scientifiques, en particulier les physiciens, pour résoudre certains problèmes conséquents. La

situation a considérablement évolué depuis, puisque le HPC est désormais utilisé intensivement dans la société moderne, de nombreuses industries recourant à la simulation numérique. L'écosystème de la recherche en HPC a donc évolué en conséquence, de nombreux acteurs privés entrant sur ce marché en rapide expansion, et obligeant les chercheurs à mieux diffuser leurs solutions logicielles pour espérer être visibles.

6.2.1 De bonnes méthodes de développement logiciel

Diffuser nos travaux via nos logiciels impose des techniques de développement adaptées. J'insiste beaucoup sur ces aspects auprès de mes collègues car diffuser un logiciel non prêt peut avoir des répercussions négatives sur notre réputation. En effet, qui n'a pas abandonné rapidement le test d'un logiciel parce qu'il ne se configurait pas facilement, ne compilait pas facilement, ou obtenait de mauvaises performances sans optimisations avancées à la main ? Nos emplois du temps surchargés nous empêchent de passer du temps à contourner tous ces petits problèmes. Et cela peut nous conduire à citer négativement un concurrent dans notre prochaine publication parce que sa solution n'est pas parfaitement portable, stable, efficace, etc.

Il faut veiller à ce que nos logiciels soient faciles à compiler, installer et utiliser. Mais il faut aussi qu'il soit facile d'en tirer de bonnes performances. Combien de logiciels proposent avec des dizaines d'options, des ordonnanceurs configurables, etc. ? Tout ça parce qu'un cas hypothétique se présenterait un jour ? En plus d'investir du temps d'ingénieur jamais rentabilisé, alors que la recherche publique est en période de disette, cette course à la configurabilité nuit en fait gravement à l'efficacité de nos logiciels. La plupart des utilisateurs n'est pas capable (ou ne souhaite pas y passer du temps) d'optimiser finement la configuration de notre logiciel pour sa plate-forme. Mais c'est pourtant cet utilisateur qu'il faut convaincre de l'intérêt de nos travaux, pas un éventuel expert qui est de toute façon déjà convaincu. Il faut donc réduire le nombre d'options de configuration au minimum et surtout veiller à ce que la configuration par défaut soit toujours satisfaisante. Autrement, on perd des utilisateurs potentiels avant même qu'ils aient eu une chance de tenter d'optimiser le logiciel pour leur plate-forme. Et là aussi, on risque d'être cité négativement. Je milite auprès de mes collègues pour répandre ces bonnes pratiques. [2]

Il faut aussi utiliser des outils de développement adaptés. L'intégration continue ou les tests de non-régression se sont démocratisés depuis plusieurs années. [3] On sait aujourd'hui comment mettre en place des suites de tests pertinentes. On peut enfin espérer la prochaine disparition des tests réseau se limitant à un ping-pong entre deux machines. J'ai travaillé avec les ingénieurs Ludovic Stordeur et Ludovic Courtès pour faire entrer ces outils dans l'équipe et espérer mettre fin aux tests trop limités, non reproductibles, etc. La portabilité et la qualité de certains de nos logiciels se sont grandement améliorées depuis. Mais il ne faut pas non plus arriver à l'autre extrême : lancer des centaines de tests trop souvent peut-être un gaspillage de temps processeur (et d'énergie, et de disponibilité des machines), surtout si on ne prend pas le temps pour en consulter régulièrement les résultats. Il faut savoir mesurer l'intérêt des différents types de tests (compilation, exécution, performance, etc.) à être lancés plus ou moins souvent.

De la même façon, on dispose d'outils modernes permettant un développement plus efficace, comme par exemple les systèmes de compilation, ou les systèmes de gestion de version. Pourtant beaucoup d'entre nous freinent encore quand on leur parle de passer de CVS ou SVN

2. *Faire des releases logicielles : pourquoi, quand et comment ?*, Les mardis du développement technologique. 5 février 2013. http://sed.bordeaux.inria.fr/les-mardis.
3. Inria a récemment mis en place la plate-forme http://ci.inria.fr dans ce but.

à GIT. La raison ? Le temps d'apprentissage, et un hypothétique utilisateur qui serait gêné s'il veut utiliser la version de développement de notre code mais n'y arrive pas à cause de ce système qu'il ne connaîtrait pas ? Mais il faut bien avouer que cet éventuel utilisateur n'existe pas, et si vraiment il existe, il y a des solutions. De toute façon, il nous faudra tous passer un jour à ces outils modernes. Autant le faire dès maintenant plutôt que de ralentir les autres développeurs, ceux qui font vraiment avancer notre logiciel. S'il est important de choyer nos utilisateurs pour qu'ils continuent à utiliser notre logiciel et propager la bonne parole, il ne faut pas non plus trop les laisser interférer dans notre cœur de métier.

Il s'agit en fait de ne pas violer la règle de moindre surprise : adopter les bons outils et les bonnes pratiques de nos pairs, en l'occurrence essentiellement ceux des logiciels libres majeurs. Ainsi, on peut espérer qu'il sera aisé pour les autres chercheurs de tester nos logiciels (car ils sont faciles à installer et à configurer), mais aussi de les modifier (sans avoir à appréhender un n-ième système de compilation étrange), et ils pourront donc contribuer à notre recherche.

Enfin, une bonne diffusion logicielle impose de correctement aborder le problème des licences. Combien de bibliothèques ont été mises en GPL par défaut sans savoir que la LGPL aurait pu avoir certains avantages pour certains publics ? Si le processus de choix de licence par les services de valorisation des instituts est parfois encore trop long et fastidieux, les chercheurs doivent s'approprier le problème en apprenant à comprendre les problématiques de diffusion et de licences.

6.2.2 De la nécessité d'une communication au-delà du public scientifique

La diffusion logicielle est très importante dans nos domaines car le HPC reste essentiellement un outil pour les autres sciences, pour les applications de simulation numérique. Contrairement à d'autres domaines, la recherche en HPC se traduit donc souvent naturellement par du code. Et ce code doit être confronté à la réalité de ces applications, testé à plus grande échelle, sur d'autres plates-formes, etc. Après tout, la reproductibilité des expériences est un aspect inhérent à la démarche scientifique. La seule publication scientifique sans diffusion du code correspondant n'est donc généralement pas suffisante. Le code doit être disponible (et être facile à utiliser comme on l'a vu en section précédente) afin que nos pairs puissent évaluer nos travaux (et les citer positivement).

Au-delà de prouver par l'implémentation qu'une idée est applicable en pratique, la diffusion du code permet également aux pairs d'améliorer nos travaux. Un logiciel de recherche qui n'est utilisé par personne signifie également que personne ne s'approprie cette recherche pour la faire avancer. La collaboration entre différents chercheurs qui apportent chacun leur petite pierre à l'édifice est pourtant cruciale à l'avancée de la science. On peut donc se demander l'intérêt de passer du temps à diffuser un logiciel s'il n'est jamais utilisé par d'autres. [4] Si on décide de diffuser un logiciel, il faut se donner les moyens de réussir cette diffusion, faute de quoi le temps investi dans le développement aura été gâché. Par contre, si on ne souhaite pas investir du temps de développement, on peut alors se limiter à une implémentation rapide de nos idées pour les valider, en espérant que cela suffise à nos pairs qui voudraient éventuellement reproduire nos résultats.

Il faut ensuite aller au-delà de la communication scientifique classique et sortir des habituels articles que publient les chercheurs. On s'adresse en fait à un public plus vaste, qui contient notamment des industriels, potentiels utilisateurs de nos résultats, et la communication doit être adaptée. De plus, avec le nombre croissant de conférences, journaux, et entreprises privées dans nos domaines, être visible est de plus en plus difficile. Monter une stratégie de communication est un métier qui s'apprend et nos collègues des services supports sont

4. On dit souvent qu'un logiciel sans utilisateur est un logiciel mort.

aussi là pour nous aider. [5]

On est malheureusement obligé de s'approcher des aspects publicitaires quand on promeut nos résultats sur des grands salons comme SuperComputing où on peut exposer des posters, faire des présentations orales, donner des tutoriels, etc. De plus, comme expliqué dans la partie précédente, de bons outils de développement sont nécessaires pour que la communication technique réussisse (partage de code, listes de diffusion, etc.). Mais ce sont de plus en plus les nouveaux moyens de communication qui révolutionnent tous ces aspects. Difficile de faire le *buzz* dans la communauté HPC sans passer par les réseaux sociaux ou sites web spécialisés qui sont les seuls moyens de toucher l'audience au-delà des scientifiques académiques. Mais il est difficile de quantifier leur impact clairement. Et, à l'inverse, il faut également savoir adapter notre veille scientifique et technologique à ces nouveaux moyens. Impossible aujourd'hui d'espérer maîtriser nos domaines de travail en ne consultant que les actes des principales conférences scientifiques.

Enfin, il faut savoir appréhender la communication abusive des sociétés privées spécialisées en HPC qui brouillent souvent les pistes en promettant monts et merveilles, par exemple avec des produits censés assurer la parallélisation automatique des codes. Cela peut réduire l'impact de nos résultats de recherche en les faisant paraître moins novateurs, mais il ne faut pas se décourager devant de telles pratiques. Si nous les chercheurs ne pouvons nous permettre d'abuser ainsi, nous avons la légitimité pour démonter leurs arguments en public puis mettre en avant les problèmes dont ils ne parlent pas et que nous, nous comprenons.

Toutes ces réflexions font parties des raisons qui me poussent à diffuser la plupart de mes travaux sous la forme de briques s'intégrant dans des bibliothèques déjà reconnues : cela garantit la diffusion de mes travaux par la réputation existante de ces bibliothèques, et donc la rentabilité du temps de développement que j'y passe.

6.2.3 La médiation scientifique en HPC, c'est possible !

Les sections précédentes illustrent la frontière ténue entre science et technologie autour de laquelle se placent nos développements logiciels. Elle nuit à l'informatique, car ce terme souffre de la confusion du grand public avec la bureautique, ce qui nous transforme régulièrement en potentiels réparateurs d'ordinateurs sous Windows. C'est notamment vrai pour les chercheurs en HPC comme moi car notre travail est effectivement très lié à la technologie des processeurs, réseaux, etc. qui évolue très vite. Et quand nous utilisons notre outil `lstopo` pour représenter de manière simple et jolie la complexité des machines, nous surfons en fait sur la facilité en utilisant la technologie pour attirer l'auditeur vers la science.

Cette perversion de notre recherche ne plaît pas à certains collègues, qui disent préférer parler uniquement de sciences. Pourtant, quand il s'agit d'aller présenter ces sciences au grand public, il n'y a plus personne ! L'excuse ? *Le HPC, c'est pas fun, on a rien de joli à montrer !* C'est pourtant faux, il suffit de voir les superbes images résultants de la simulation numérique pour se convaincre que le HPC peut produire de jolies choses. Là aussi, on pervertit notre recherche en utilisant les applications du HPC pour présenter notre travail dans les couches basses. Est-ce vraiment un problème ? Si on pose la question aux physiciens qui conçoivent ces simulations numériques, ils pourraient également répondre que leur science, c'est des équations, pas l'affichage des résultats. Les jolis résultats seraient donc réservés aux seuls graphistes ?

Évidemment, on ne peut pas utiliser cette excuse pour ignorer notre mission de faire aussi de la médiation scientifique. Nous avons également pour rôle d'aller vers le grand public pour justifier l'investissement de la société dans nos travaux, vers le citoyen pour lui donner les clefs de compréhension des enjeux de la société numérique, vers les jeunes pour susciter des

5. Malheureusement encore trop de chercheurs ne comprennent pas les métiers de la communication.

vocations, etc. On se limite encore trop souvent à l'aspect péjoratif du terme *vulgarisation*, qui peut effectivement dénaturer notre recherche, ce qui explique la réticence de certains collègues. Mais la médiation scientifique va bien au-delà d'une simple traduction, c'est un échange avec le public, qui nous oblige à prendre du recul sur nos travaux, pour comprendre des points de vue très différents, comme par exemple des questions éthiques. [6] Parler de la technologie ou des applications de nos travaux, ce n'est après tout qu'une introduction, une présentation du contexte. Si cela peut également servir à attirer l'attention de l'auditeur pour engager la discussion, pourquoi hésiter ? Il sera plus facile ensuite de lui passer nos messages scientifiques. Et ceci peut de toute façon être réutilisé pour convaincre des industriels ou des scientifiques d'autres domaines.

J'ai été amené depuis plusieurs années, dans ma mission de chargé de médiation scientifique de mon centre de recherche, à travailler sur ces aspects. Une de mes satisfactions personnelles est que nous sommes parvenus à retirer l'étiquette collée sur le front du HPC qui disait qu'il n'y avait rien de joli ou d'intéressant à présenter au grand public. La complexité des consoles de jeu a par exemple permis de présenter le calcul parallèle sur plates-formes hétérogènes [7] tandis que la recette des gaufres nous a permis d'expliquer les algorithmes parallèles à des collégiens. [8] (l'ensemble de mes activités de médiation scientifique est résumé en annexe A.4).

6.3 Combler le vide entre les développeurs système et le HPC ?

Sous l'impulsion de Loïc Prylli qui a encadré le début de ma thèse, j'ai mené toute ma carrière à l'interface entre le système d'exploitation et les supports exécutifs pour le HPC. Les difficiles interactions entre ces deux mondes m'ont toujours frappé. Mais plus généralement, le cloisonnement entre les différentes communautés dans le HPC reste important. On s'en rend notamment compte dans les organismes de standardisation.

6.3.1 Apporter des compétences aux organismes de standardisation

L'équipe Runtime de Bordeaux travaille à la fois sur de nouveaux modèles de programmation et sur des améliorations des implémentations de standards existants (notamment MPI et OpenMP). J'ai donc été amené à participer à des réunions d'organismes de standardisation (voir en annexe A.5.5), en particulier le MPI Forum [9] où je suis un des représentants d'Inria.

J'ai été plusieurs fois frappé du manque d'interaction entre les différents organismes. Par exemple, pendant la conception de MPI 3.0, des propositions ont été faites pour faciliter l'interaction entre MPI et OpenMP. Il est vrai que ces modèles hybrides peinent toujours à être implémentés efficacement. Mais ces propositions nous ont paru peu matures techniquement, notamment parce que les membres du MPI Forum connaissent en général mal les contraintes imposées par la sémantique et les implémentations des threads, ou au mieux ne les connaissent que sur certaines plates-formes. Heureusement, nous avons pu faire en sorte que ces propositions ne soient pas standardisées.

Un autre exemple est l'absence de personnes connaissant les pilotes matériels dans le MPI Forum. Si les principaux constructeurs du HPC sont membres du forum, peu y participent activement. Or, quand il s'est agi de concevoir l'interface RMA (*Remote Memory Access*) de MPI

6. Le gaspillage énergétique des supercalculateurs est souvent cité dans les échanges autour du HPC.
7. *Quand votre console de jeu se prend pour une calculatrice*. Fête de la Science 2010 et 2011.
8. *Faire des gaufres en parallèle, seriez-vous plus efficace que votre ordinateur ?* Fête de la Science 2012, avec Bertrand Putigny et François Tessier.
9. http://www.mpi-forum.org.

3.0, il a fallu vérifier les contraintes de ces pilotes. Et j'ai été surpris de voir qu'on insistait sur ma présence aux réunions du forum car j'étais le seul à connaître le noyau Linux et ses contraintes, grâce à mes travaux chez Myricom puis sur Open-MX et KNEM. On reviendra d'ailleurs sur l'interaction entre le HPC et le noyau dans la section suivante.

Enfin, les propositions de notre équipe à l'ARB qui standardise OpenMP [10] ont révélé un certain malaise de cet organisme vis-à-vis de l'ordonnancement dynamique de threads. Des collègues barcelonais nous ont plus tard confirmé que l'ARB était constitué de personnes du monde des compilateurs, qui maîtrisent donc essentiellement la compilation et l'ordonnancement statique. Elles ne souhaitaient donc pas standardiser des idées qui leur imposeraient du travail dans le support exécutif, car elles n'y sont pas habituées. Malheureusement pour OpenMP, ce cloisonnement entre le monde de la compilation et celui des supports exécutifs risque d'être de plus en plus problématique sur les futures plates-formes très complexes. Il va devenir de plus en plus important de prendre les décisions dynamiquement.

Mon chef d'équipe, Raymond Namyst, insiste depuis plusieurs années sur la nécessité pour nous d'assister aux réunions de standardisation. Dans le MPI Forum, notre compétence est connue et respectée, nous avons pu y faire passer la bonne parole. Dans l'OpenMP, le résultat est tout autre. Nous avons publié plusieurs fois dans le workshop IWOMP [WI1, WI11], [15] mais notre logiciel ForestGOMP était très difficile à utiliser (ce qui renvoie à la section 6.2.1). Si on souhaite changer les choses, il faudra être mieux reconnus par les membres de l'ARB.

6.3.2 Comment travailler avec les gens du noyau Linux ?

J'ai également été frappé par les mauvaises interactions entre le HPC et la communauté du noyau Linux. Comme évoqué en section 2.1.1, le HPC a longtemps été considéré comme une bande de hackers visant la performance au détriment de la stabilité ou de la portabilité. Les choses se sont améliorées doucement depuis l'intégration des pilotes InfiniBand dans le noyau, mais il reste très difficile d'ajouter des fonctionnalités spécifiques au HPC. J'y vois trois raisons. [11]

Premièrement, le HPC reste un **petit marché de niche** avec des besoins très spécifiques. Que peut-on accepter de faire pour obtenir un gain de 100% de performance en HPC ? Certainement pas perdre 5% de performance pour les millions d'autres utilisateurs. Rendre le code beaucoup plus compliqué pour ajouter un cas spécifique au HPC ? C'est souvent l'excuse des développeurs noyau pour rejeter une proposition. [12] Il faut enfin noter que certaines fonctionnalités nécessaires au HPC (comme celles listées en section 2.1.2) ont en fait été ajoutées car d'autres utilisateurs en avaient besoin : les pilotes de cartes graphiques et la virtualisation, qui sont beaucoup plus largement utilisés que le HPC.

Deuxièmement, les développeurs noyau sont habitués à n'accepter de discuter que de fonctionnalités qui ont été **proprement implémentées et testées**. Le temps que cela demande est difficilement compatible avec le travail de chercheur où on se limite souvent à une implémentation rapide pour prouver l'intérêt d'une idée. [13] D'ailleurs, avec l'augmentation de l'importance du HPC dans la société numérique, certaines entreprises commencent à voir l'intérêt de contribuer au HPC, ce qui explique pourquoi Red Hat et IBM ont été impliqués dans l'intégration de fonctionnalités citées au point précédent.

10. *The OpenMP Architecture Review Board*, `http://openmp.org/wp/about-openmp/`.
11. J'ai expliqué ces constats lors d'une présentation aux rencontres mondiales du logiciel libre en 2011. *Linux vs HPC : Life (and death) of strange features*.
12. L'interface `ummunot` visant à simplifier la gestion des caches d'enregistrement mémoire a par exemple été refusée ainsi : `http://www.open-mpi.org/community/lists/users/2012/11/20671.php`.
13. `https://lkml.org/lkml/2010/10/12/178` illustre la réputation de la recherche académique auprès de certains développeurs noyau.

Non seulement le développement noyau est difficile, mais en plus il faut veiller attentivement à ne pas dégrader les performances hors HPC. De plus, le processus de soumission et vérification du code est long et parfois peu courtois [14], ce qui peut décourager certains chercheurs. On peut donc raisonnablement se demander l'intérêt pour un chercheur de passer du temps à intégrer son travail dans le noyau Linux. Si la nouvelle fonctionnalité touche au cœur du noyau, elle ne peut pas être chargée dynamiquement comme un module externe, son intégration au noyau standard est nécessaire à sa diffusion. J'ai effectivement intégré de nombreuses contributions (tous mes patchs sont détaillés en annexe A.2.2). Concernant mes modules externes Open-MX et KNEM, je n'ai pas insisté pour les faire intégrer car ils peuvent facilement être installés et chargés dynamiquement, et les administrateurs HPC sont souvent habitués à ce genre de (petites) contraintes.

Il faut aussi noter que si le domaine est concurrentiel, la première solution intégrée va freiner la concurrence, qui devra souvent améliorer la première plutôt que d'intégrer aussi sa solution alternative. C'est probablement la raison pour laquelle la popularité de KNEM va diminuer à l'avenir : le noyau contient une fonctionnalité similaire (comme on l'a vu en section 3.4.3) et les utilisateurs vont avoir tendance à la privilégier même si elle est très limitée par rapport à KNEM.

Troisièmement, il y a **très peu de personnes maîtrisant à la fois le HPC et le noyau Linux**. Cela conduit par exemple à mépriser la sécurité au profit des performances comme on l'a vu à propos de LiMIC en section 3.4.3. Cette faible densité a en fait été agréable pour moi car mes travaux sur Open-MX et KNEM trouvent difficilement de la concurrence dans la communauté académique et ont donc pu y percer plus facilement. Par contre, cela m'a rendu difficile le recrutement de jeunes chercheurs car les compétences requises pour travailler sur ces projets sont rares. C'est aussi pour cela que je veille à conserver la responsabilité du cours et des TD de système d'exploitation à l'ENSEIRB-MATMECA : nos étudiants sont trop rarement formés au développement bas niveau.

6.3.3 Linux est-il adapté au HPC ?

J'ai beaucoup parlé de portabilité dans ce manuscrit, en particulier dans les chapitres 2 et 3 et on pourrait pourtant me reprocher de n'avoir travaillé que sur Linux. Tout d'abord, la portabilité dont je parlais était essentiellement liée au matériel : les fonctionnalités que j'ai tenté de généraliser n'étaient préalablement pas portables car disponibles uniquement dans les pilotes de certains réseaux rapides. Ensuite, il ne faut pas oublier que Linux reste extrêmement majoritaire en HPC, il est utilisé dans 482 des 500 supercalculateurs les plus puissants au monde en Novembre 2013 (Top500 [70]). Cette hégémonie a des inconvénients, mais elle me permet de justifier que mes travaux ne ciblent que ce système d'exploitation afin d'avoir une vaste audience. Par ailleurs, toutes mes idées et implémentations logicielles étant publiées, leur portage vers un autre système n'est pas difficile.

On peut cependant raisonnablement se demander si Linux convient réellement au HPC, où s'il n'est qu'un choix par défaut. Certains problèmes de Linux ont souvent été cités pour justifier l'idée d'utiliser un autre système :

— Les mécanismes de gestion de mémoire virtuelle, en particulier le *Swap* est très problématique pour les communications haute performance (que ce soit les réseaux rapides ou les solutions logicielles comme Open-MX et KNEM) ;
— L'ordonnanceur ne place pas correctement les tâches à exécuter et casse les affinités entre tâches ou entre tâches et caches.

14. Linus, Sarah and the Linux Civil Code. http://www.linuxinsider.com/story/78536.html.

L'idée derrière tout cela est que Linux est un système généraliste qui essaie de bien supporter tous les cas, sans en supporter aucun parfaitement. Un système d'exploitation est obligé de faire des compromis pour s'adapter à ce que les applications lui demandent, en essayant de comprendre au mieux ce qu'elles souhaitent vraiment. Mais en HPC, on souhaite souvent désactiver cette pseudo-intelligence imparfaite et faire directement au mieux ce dont on a besoin, avec les meilleures performances. C'est un vieux problème auquel je me heurtais déjà pendant mon post-doctorat chez Myricom lors du développement des cartes du support logiciel pour les cartes Myri-10G [RA1]. Le verrouillage mémoire et le placement des processus est un moyen de contourner les problèmes sus-cités mais cela reste imparfait.

Cela illustre d'ailleurs bien pourquoi les développeurs noyau n'aiment pas trop le HPC : ils essaient de concevoir des heuristiques intelligentes pour s'adapter aux différents cas, et la communauté HPC essaie de les éviter. Plus généralement, de nombreux développeurs noyau pensent qu'on peut tout faire automatiquement dans le noyau sans avoir besoin de laisser l'application faire les choses elle-même. [15] Ce pourrait être vrai si l'application avait de bons moyens d'indiquer ses besoins, mais ce n'est pas le cas : il est par exemple impossible dans Linux de spécifier les affinités entre tâches, l'ordonnanceur ne peut donc pas en tenir compte dans ses décisions de placement. Il pourrait tenir compte des compteurs de performance matériels pour l'aider, mais ces informations ne peuvent pas être traduites en affinités précises. Cette situation semble malheureusement difficile à changer tant que la réputation et l'importance du HPC auprès des développeurs noyau ne se sera pas améliorée. En attendant, on doit continuer à essayer de contourner les heuristiques imparfaites du noyau et espérer avoir dans nos bibliothèques toutes les informations utiles, ce qui n'est pas toujours facile comme on l'a vu en section 4.2.4.

6.3.4 Quel système d'exploitation pour le HPC du futur ?

Si Linux n'est pas parfait pour le HPC, une idée consiste à le remplacer par un système d'exploitation minimaliste qui n'aurait pas tous ces défauts. En pratique, cela a longtemps été impossible à grande échelle car les administrateurs de supercalculateurs ne peuvent pas installer un système d'exploitation quelconque sans prendre de grands risques pour la sécurité des utilisateurs et de leurs données. La virtualisation a résolu ce problème en permettant aux utilisateurs d'installer ce qu'ils souhaitent sans de tels risques. Malheureusement, et malgré toute la communication contradictoire, ce n'est pas sans un impact sur les performances, notamment pour les entrées-sorties.

L'idée de construire un système spécialisé pour le HPC est en fait un vieux serpent de mer. Elle ne s'est jamais matérialisée par un système stable qu'on aurait pu envisager de déployer sur de nombreuses plates-formes. Des travaux ont visé spécifiquement les machines BlueGene avec le développement de ZeptoOS [67] qui a servi de plate-forme d'expérimentation pour mettre en évidence les lacunes des systèmes d'exploitation à l'approche du petascale [10]. Mais ce travail restait très spécifique à une architecture très différente de la majorité des supercalculateurs.

Le projet Argo [5] a récemment été annoncé pour formaliser une collaboration entre de nombreuses équipes américaines dans le but de développer un système d'exploitation pour l'Exascale à l'horizon 2020. C'est beaucoup plus ambitieux que ZeptoOS puisque la portabilité sur différentes architectures est cette fois visée. On peut toutefois se demander si une telle initiative peut fonctionner. Définir un nouveau système portable représente un énorme travail. Et même si l'accent va être mis sur le HPC, il restera tout de même énormément de fonction-

15. Les développeurs de Mac OS X ont d'ailleurs un comportement similaire en ne permettant pas à l'utilisateur de binder ses processus.

nalités à implémenter. On risque donc de réinventer toute une partie de Linux, mais avec une communauté de développeurs beaucoup moins grande. La complexité des mécanismes des systèmes d'exploitation modernes ne peut pas être négligée. [16] Qu'en sera-t-il de la capacité de ces personnes à porter le système sur différentes architectures et d'en assurer la maintenance à long terme ? Une solution plus satisfaisante consisterait peut-être en la modification de Linux (pour en enlever certaines contraintes pour le HPC), puis son intégration dans un système (distribué) gérant les supercalculateurs. Difficile de savoir dès maintenant ce que Argo proposera à terme.

Par ailleurs, comment concevoir un système alors qu'on ne sait pas quel modèle de programmation sera utilisé pour l'Exascale, ni quelle architecture ? Comme je l'explique dans l'article de vulgarisation scientifique [MS1], gérer des millions de tâches sur des millions de processeurs est difficile. Le système est l'interface entre le matériel et les logiciels, comment en concevoir un si on ignore de quoi seront faites les deux couches qu'il reliera ? J'ai tendance à penser que les supercalculateurs exaflopiques continueront plutôt à utiliser des environnements logiciels proches de ce que nous avons actuellement.

6.4 À quel matériel et innovations logicielles s'attendre ?

6.4.1 De nombreux petits cœurs ?

Il semble désormais assez probable que les supercalculateurs du futur seront constitués de très nombreux petits cœurs, voie ouverte par les architectures BlueGene. L'accélérateur Xeon Phi récemment lancé par Intel reprend un modèle assez similaire à BlueGene, avec des cœurs x86 simplifiés mais disposant d'unités de calcul flottant vectoriel très larges (SIMD). NVIDIA abuse du terme cœur en parlant de milliers de *CUDA cores* dans ses cartes graphiques, mais on peut finalement considérer qu'elles contiennent quelques dizaines de cœurs SIMD (les *shared-multiprocessors*).

L'avantage de ces architectures est que ces cœurs plus simples sont beaucoup plus économes que les gros cœurs généralistes x86 [17]. Ces derniers ont tout de même un avantage sur les codes séquentiels, ce qui tend à penser que les architectures mixtes vont continuer à être utiles. Après tout, il ne faut pas oublier que certains codes ne peuvent pas être vectorisés. ARM propose d'ores et déjà des processeurs mélangeant quelques petits cœurs économes et quelques gros cœurs gourmands et puissants [29]. C'est dans une certaine mesure comparable aux serveurs actuels mélangeant CPU et accélérateurs. Reste à nos logiciels à être capables d'exploiter intelligemment cette complexité.

Mais qui va décider du futur des architectures HPC ? Probablement le marché grand public. En effet, le HPC est un marché trop petit pour rentabiliser des investissements spécifiques. Intel a conçu le Xeon Phi pour le HPC, mais il en vendra difficilement des millions par an et aura donc du mal à compenser le coût de la conception. Par contre, les vastes marchés pour les serveurs ou les jeux vidéos permettent de rentabiliser les investissements dans les processeurs généralistes ou dans les cartes graphiques. Le HPC ne peut qu'utiliser des technologies rentabilisées sur de tels marchés. Or, avec la chute annoncée des ventes d'ordinateurs, le marché du futur pour les processeurs est probablement celui de l'embarqué, des smartphones, des tablettes, etc. où règnent ARM, Intel Atom, etc. Le futur du HPC est peut-être là ? À moins

16. Il se dit souvent que personne ne comprend plus la gestion mémoire de Linux dans sa globalité.
17. Le Green500 (`http://www.green500.org/`) qui classe les supercalculateurs selon leur efficacité énergétique montre que les clusters de processeurs généralistes atteignent difficilement 1 Gigaflop/s/W tandis que les architectures BlueGene ou à accélérateurs dépassent souvent le double.

qu'Intel ne parvienne à convaincre les serveurs hors-HPC (Big Data ?) d'utiliser des accélérateurs ?

6.4.2 La non-cohérence de cache, la menace fantôme ?

Avec l'augmentation continue du nombre de cœurs, une question revient souvent : les architectes matériels vont-ils pouvoir assurer la cohérence de cache encore longtemps ? Le maintien de cette cohérence coûte en effet très cher, comme on l'a évoqué en section 4.3, non seulement au niveau applicatif, mais aussi en terme de transistors dans les processeurs. Pour le moment, cette barrière continue à être repoussée. Le Xeon Phi est capable d'assurer la cohérence entre les caches de 61 cœurs, tandis que SGI est capable de construire des machines regroupant 256 sockets (3072 cœurs) en mémoire partagée. [18] Et certains collègues pensent que nous serons capables de continuer ainsi pendant longtemps [50].

En supposant que la cohérence de cache deviendra à moyen terme impossible à supporter matériellement, à quel niveau la brisure pourrait-elle apparaître ? Avec la remontée annoncée de l'interface réseau dans le processeur, on peut se demander si les machines du futur continueront à être multiprocesseur. Si l'accès réseau est très performant, pourquoi s'embêter avec un bus mémoire inter-processeur de plus en plus cher à concevoir ? Dans tous les cas, en dehors du cas particulier SGI, les serveurs à plus de 2 processeurs sont de plus en plus rares. Le degré de parallélisme explose uniquement à l'intérieur des processeurs. Doit-on donc s'attendre à la sortie de processeurs dont les cœurs ne sont plus cache cohérents ? Les applications parallèles existantes, en particulier les programmes multithreadés, ne pourraient plus fonctionner. Casser ainsi la compatibilité avec les utilisateurs existants est inenvisageable pour les constructeurs. La transition devra se faire doucement, avec par exemple plusieurs îlots de cohérence dans chaque processeur ? Les applications existantes pourraient donc continuer à s'exécuter en parallèle dans chaque îlot, mais devraient être adaptées pour utiliser l'ensemble de la machine.

Comment alors modifier nos programmes pour contourner la non-cohérence de cache ? La solution simple consiste à s'assurer que les données partagées sont manipulées directement en mémoire centrale, sans cache intermédiaire. Certaines zones peuvent ainsi être marquées comme totalement non cachables dans les processeurs. Mais l'état de ces zones ne doit en fait être assuré qu'au moment du transfert entre tâches, inutile de ralentir les autres accès. On peut donc attendre le premier accès concurrent avant de forcer leur éviction du cache local. C'est en fait similaire à certaines instructions existant déjà sur les processeurs actuels, mais il faudrait en préciser les garanties.

Mais quitte à confier aux applications ce genre d'opérations, ne devrait-on pas plutôt leur laisser gérer les caches locaux logiciellement, comme des *scratchpad* ? C'est ce que proposait le processeur Cell [75] pour les 256 ko de chaque cœur. Bertrand Putigny qui a expérimenté l'idée sur Cell et Cyclops64 [32] milite souvent dans cette direction. L'application doit transférer explicitement les données dans cette mémoire locale avant de les utiliser. Elle gère donc la cohérence elle-même intégralement, ce qui simplifie le matériel et donne plus de pouvoir aux développeurs, mais aussi plus de travail.

Intel a développé le processeur prototype SCC (*Single Chip Cloud Computer* [51]) pour permettre aux chercheurs de développer en amont des logiciels adaptés à ces architectures. Il semble qu'une interface de passage de messages similaire à MPI [19] (mais avec des transferts de données et synchronisations adaptés) pourrait être un bon moyen d'exploiter une telle plateforme. Cependant, ce résultat reste très largement à confirmer car le SCC ne fournit presque

18. SGI Altix UV http://www.sgi.com/products/servers/uv/.
19. La programmation du SCC repose souvent sur l'interface RCCE qui est similaire à un MPI minimaliste.

pas de fonctionnalités pour gérer la non-cohérence de cache : il faut par exemple remplir manuellement un cache pour être sûr qu'une donnée n'y est plus. On ignore ce que les constructeurs implémenteront dans les éventuels futurs processeurs sans cohérence de cache, on se doit d'espérer mieux que le SCC, mais il est malheureusement difficile pour nous d'influencer ces choix. Les petits constructeurs comme Kalray (qui propose le processeur MPPA à 256 cœurs non cache-cohérents [21]) sont plus faciles à influencer à travers des collaborations mais leur avenir est malheureusement trop incertain.

Une question qui se posera également est la répartition du ou des systèmes d'exploitation. Les systèmes modernes conservent une vue globale de la machine, mais doivent de plus en plus souvent prendre des décisions locales, par exemple pour l'ordonnancement, car les statistiques guidant ces choix sont trop coûteuses à analyser au niveau global. On fait plus du gouvernement hiérarchique que du management individuel des ressources. Dans une machine sans cohérence de cache, une gestion globale sera encore plus difficile. On pourrait donc avoir un système d'exploitation par îlot de cohérence ? Cela fait aussi partie des réflexions intéressantes pour le projet Argo évoqué en section précédente.

6.4.3 Des accès mémoire de plus en plus critiques

En parallèle de la potentielle difficulté de maintenir la cohérence de cache entre tous les cœurs d'une machine, un problème va se présenter : celui du *memory-wall* [76]. En effet, la puissance des processeurs augmente plus rapidement que celle de la mémoire et les accès mémoire doivent donc être de plus en plus optimisés pour éviter de ralentir les calculs. Si les architectes matériels ont réussi à masquer cet effet en ajoutant des caches, en réordonnant les instructions, etc., sauront-ils combler le fossé qui continue de s'élargir ?

Mais plus que la vitesse, c'est la consommation d'énergie qui pourrait être le critère limitant prioritaire. On s'attend en effet à ce qu'une opération arithmétique sur les futures machines Exascale coûte de l'ordre 1 pJ par octet tandis qu'un accès mémoire coûtera environ 100 pJ par octet. À ce prix, impossible d'envisager d'effectuer une seule opération sur chacun des éléments de données. Il va falloir s'assurer que chaque accès mémoire est rentabilisé, c'est-à-dire qu'il est utilisé pour de nombreuses opérations arithmétiques. Et il deviendra parfois plus intéressant de recalculer un résultat intermédiaire plutôt que d'aller le chercher en mémoire.

Ce problème semble a priori plutôt se poser pour nos collègues de la compilation et de la génération de noyau de calcul. Il va également concerner les domaines du *Big Data* où les volumes de données analysés sont gigantesques. Pour les supports exécutifs et les systèmes d'exploitation, le phénomène va également impliquer que la localité des données va continuer à être très importante : si on doit accéder à une donnée en mémoire, elle a intérêt à être le plus proche possible pour ne pas faire exploser la facture énergétique.

6.4.4 Quels défis pour les chercheurs en HPC ?

Avec la taille croissante des supercalculateurs, de nombreux collègues s'intéressent aux problématiques désormais critiques telles que le passage à l'échelle ou la tolérance aux pannes. La localité va également devenir de plus en plus importante et il nous faudra être capable de la gérer à tous les niveaux de la pile logicielle. Non seulement un accès à une mémoire distante pourrait être plus lent, mais en plus il pourrait ne pas être cohérent si les caches n'ont pas été préalablement synchronisés. On peut tout d'abord appréhender ces contraintes de localité de manière statique, comme ce que fournit hwloc en se basant sur les caractéristiques matérielles, mais il faut les quantifier pour aider l'application à les utiliser, par exemple avec des travaux comme ceux de Bertrand Putigny (section 4.3). Ensuite on peut attaquer le problème de ma-

nière dynamique, comme par exemple le logiciel StarPU développé dans l'équipe qui utilise l'historique des exécutions précédentes pour s'adapter dynamiquement à la localité.

Sur de très grandes architectures, une approche purement dynamique risque de demander trop de temps d'analyse pour prendre les décisions. Une approche statique ne peut pas en revanche facilement s'adapter aux évolutions de l'application. Il faut donc combiner les deux approches en prenant quelques décisions statiques entrecoupées d'adaptations dynamiques. Des outils comme les compteurs de performance peuvent être un moyen de savoir quand il faut passer d'une approche à l'autre. Cela tend tout de même à penser que le fossé entre performances théoriques annoncées par le constructeur et performances effectivement observées dans nos applications va continuer à s'élargir, à moins d'avoir le temps d'adapter précisément notre code à chaque plate-forme.

De manière plus générale, simuler ou simplement comprendre le comportement des plates-formes parallèles et a fortiori des applications qu'elles exécutent, va devenir de plus en plus difficile. On l'étudie dans le cadre du projet ANR Songs et on sait déjà que le comportement d'une application à l'échelle dépend d'énormément de paramètres matériels et logiciels. Il y a de plus en plus de fonctionnalités avancées, intriquées et cachées dans le matériel, qui rendent notre travail difficile, que ce soit dans les processeurs [20] ou dans le système d'entrées-sorties comme les cartes réseau [21]. Un collègue d'Intel nous confirmait d'ailleurs il y a quelques années que nous aurions encore du travail pendant longtemps car même eux comprennent de moins en moins ce qui se passe dans leurs processeurs !

Enfin, il nous reste à définir les modèles de programmation du futur. Ne serait-ce qu'en terme de programmation réseau, on n'est toujours pas parvenu à définir une interface générique et portable comme on l'a vu en section 2.4.4. Il reste étonnamment beaucoup à faire pour permettre aux applications d'utiliser facilement et efficacement les technologies réseau qui semblent pourtant avoir atteint une certaine maturité après l'euphorie des années 2000. Mais c'est surtout au niveau du modèle de programmation que le travail reste gigantesque. Les modèles hybrides à la MPI+OpenMP peinent à convaincre, notamment parce que les implémentations ne sont pas à la hauteur (problème de la poule et de l'œuf). Les modèles MPI+X basées sur les accélérateurs souffrent de l'évolution trop rapide des technologies matérielles et de leurs interfaces de programmation (CUDA, OpenCL, OpenACC, etc.). Enfin les nouveaux modèles à base de PGAS ou de graphes de tâches sont très fragmentés, avec de nombreuses interfaces et implémentations différentes semblant peu ou pas converger. De plus, l'inertie des codes en HPC me conduit à penser que le paysage des modèles de programmation pour le HPC n'évoluera pas significativement à court terme. On peut donc assez sereinement continuer à travailler sur des modèles actuels en ayant confiance dans leur utilité future.

20. Les politiques de gestion des caches deviennent de plus en plus compliquées [36]. Il semble que les processeurs Intel utilisent effectivement de telles politiques adaptatives depuis la micro-architecture Ivy Bridge (`http://blog.stuffedcow.net/2013/01/ivb-cache-replacement/`).

21. Les cartes réseau, que ce soit Ethernet ou InfiniBand, peuvent implémenter plus ou moins matériellement certaines fonctionnalités, rendant le comportement et les performances très variables d'un modèle à l'autre.

Bibliographie

[1] Tarek S. ABDELRAHMAN et Thomas N. WONG. « Distributed array data management on NUMA multiprocessors ». In : *Proceedings of SHPCC*. 1994, p. 551–559.

[2] Anant AGARWAL et al. « APRIL : A Processor Architecture for Multiprocessing ». In : *Proceedings of the 17th Annual International Symposium on Computer Architecture*. 1990, p. 104–114.

[3] Thomas E. ANDERSON, David E. CULLER et David A. PATTERSON. « A Case for NOW (Networks of Workstations ». In : *IEEE Micro* 15.1 (fév. 1995), p. 54–64.

[4] *Coraid : The Linux Storage People*. http://www.coraid.com.

[5] *Argo : An exascale operating system*. URL : http://www.mcs.anl.gov/project/argo-exascale-operating-system.

[6] Scott ATCHLEY et al. « The Common Communication Interface (CCI) ». In : *19th Annual IEEE Symposium on High-Performance Interconnects*. Santa Clara, CA, août 2011.

[7] Cédric AUGONNET et al. « StarPU : A Unified Platform for Task Scheduling on Heterogeneous Multicore Architectures ». In : *Concurrency and Computation : Practice and Experience, Special Issue : Euro-Par 2009* 23 (2 fév. 2011), p. 187–198.

[8] Olivier AUMAGE et al. « High-Performance Multi-Rail Support with the NewMadeleine Communication Library ». In : *Proceedings of the Sixteenth International Heterogeneity in Computing Workshop (HCW 2007), held in conjunction with IPDPS 2007*. Long Beach, CA, mar. 2007.

[9] Amnon BARAK, Ilia GILDERMAN et Igor METRIK. « Performance of the Communication Layers of TCP/IP with the Myrinet Gigabit LAN ». In : *Computer Communication* 22.11 (juil. 1999).

[10] Pete BECKMAN et al. « Operating system issues for petascale systems ». In : *ACM SIGOPS Operating Systems Review* 40.2 (avr. 2006), p. 29–33. ISSN : 0163-5980. DOI : 10.1145/1131322.1131332.

[11] Nanette J. BODEN et al. « Myrinet : A Gigabit-per-Second Local Area Network ». In : *IEEE Micro* 15.1 (1995), p. 29–36.

[12] Timothy BRECHT. « On the Importance of Parallel Application Placement in NUMA Multiprocessors ». In : *Proceedings of the Fourth Symposium on Experiences with Distributed and Multiprocessor Systems (SEDMS IV)*. San Diego, CA, sept. 1993.

[13] Ron BRIGHTWELL, Trammell HUDSON et Kevin PEDRETTI. « SMARTMAP : Operating System Support for Efficient Data Sharing Among Processes on a Multi-Core Processor ». In : *Proceedings of the ACM/IEEE Conference on High Performance Networking and Computing, SC 2008*. Austin, TX : ACM Press, nov. 2008.

[14] François BROQUEDIS. « De l'exécution d'applications scientifiques OpenMP sur architectures hiérarchiques ». Thèse de doct. 351 cours de la Libération — 33405 TALENCE cedex : Université Bordeaux 1, déc. 2010. URL : http://www.theses.fr/2010BOR14190.

[15] François BROQUEDIS et al. « Scheduling Dynamic OpenMP Applications over Multicore Architectures ». In : *International Workshop on OpenMP (IWOMP)*. West Lafayette, IN, mai 2008. URL : http://hal.inria.fr/inria-00329934.

[16] D. BUNTINAS, G. MERCIER et W. GROPP. « Data Transfers between Processes in an SMP System : Performance Study and Application to MPI ». In : *Parallel Processing, 2006. ICPP 2006. International Conference on* (août 2006), p. 487–496. ISSN : 0190-3918. DOI : 10.1109/ICPP.2006.31.

[17] Darius BUNTINAS, Guillaume MERCIER et William GROPP. « Implementation and Evaluation of Shared-Memory Communication and Synchronization Operations in MPICH2 using the Nemesis Communication Subsystem ». In : *Parallel Computing, Selected Papers from EuroPVM/MPI 2006* 33.9 (sept. 2007), p. 634–644.

[18] H. K. Jerry CHU. « Zero-Copy TCP in Solaris ». In : *Proceedings of the USENIX Annual Technical Conference*. San Diego, CA, 1996, p. 253–264.

[19] Giuseppe CIACCIO et Giovanni CHIOLA. « GAMMA and MPI/GAMMA on GigabitEthernet ». In : *Proceedings of 7th EuroPVM-MPI conference*. Balatonfured, Hongrie, sept. 2000, p. 129–136.

[20] David D. CLARK et al. « An Analysis of TCP Processing Overhead ». In : *IEEE Communications Magazine* 27 (1989), p. 23–29.

[21] Benoît Dupont de DINECHIN et al. « A Distributed Run-Time Environment for the Kalray MPPA®-256 Integrated Manycore Processor ». In : *Procedia Computer Science* 18 (2013), p. 1654–1663. ISSN : 1877-0509. DOI : 10.1016/j.procs.2013.05.333.

[22] Jack EDGE. *Receive flow steering*. http://lwn.net/Articles/382428/. Avr. 2010.

[23] Alexandre E. EICHENBERGER et al. « The Design of OpenMP Thread Affinity ». In : *OpenMP in a Heterogeneous World - 8th International Workshop on OpenMP (IWOMP 2012)*. Rome, Italy : Springer, juin 2012.

[24] *FCoE (Fibre Channel over Ethernet)*. http://www.fcoe.com.

[25] Agner FOG. *Instruction tables Lists of instruction latencies, throughputs and micro-operation breakdowns for Intel, AMD and VIA CPUs*. http ://www.agner.org/optimize/. 2011.

[26] Matteo FRIGO et al. « Cache-Oblivious Algorithms ». In : *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*. New York City, NY : IEEE Computer Society, oct. 1999, p. 285.

[27] Patrick GEOFFRAY, Loïc PRYLLI et Bernard TOURANCHEAU. « BIP-SMP : high performance message passing over a cluster of commodity SMPs ». In : *Proceedings of the 1999 ACM/IEEE conference on Supercomputing*. Portland, OR, nov. 1999.

[28] Jorge GONZÁLEZ-DOMÍNGUEZ et al. « A Benchmark Suite for Autotuning on Multicore Clusters ». In : *Proceedings of 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS'10)*. Atlanta, GA : IEEE Computer Society Press, avr. 2010.

[29] Peter GREENHALGH. *Big.LITTLE Processing with ARM Cortex-A15 & Cortex-A7 : Improving Energy Efficiency in High-Performance Mobile Platforms*. ARM Whitepaper. Sept. 2011. URL : http://www.arm.com/files/downloads/big_LITTLE_Final_Final.pdf.

[30] Leonid GROSSMAN. « Large Receive Offload Implementation in Neterion 10GbE Ethernet Driver ». In : *Proceedings of the Linux Symposium (OLS2005)*. Ottawa, Canada, juil. 2005, p. 195–200.

[31] Torsten HOEFLER et al. « The scalable process topology interface of MPI 2.2 ». In : *Concurrency and Computation : Practice and Experience* 23.4 (2011), p. 293–310.

[32] Ziang HU et al. *Programming Experience on Cyclops-64 Multi-Core Chip Architecture*. 2006.

[33] C.J. HUGHES et al. « Rsim : simulating shared-memory multiprocessors with ILP processors ». In : *Computer* 35.2 (2002), p. 40–49. ISSN : 0018-9162. DOI : 10.1109/2.982915.

[34] Joshua HURSEY, Jeffrey M. SQUYRES et Terry DONTJE. « Locality-aware Parallel Process Mapping for Multi-Core HPC Systems ». In : *International Conference on Cluster Computing (IEEE Cluster)*. Austin, Texas : IEEE Computer Society Press, sept. 2011, p. 527–531.

[35] *InfiniBand architecture specifications*. InfiniBand Trade Association. http://www.infinibandta.org. 2001.

[36] Aamer JALEEL et al. « High performance cache replacement using re-reference interval prediction (RRIP) ». In : *SIGARCH Computer Architecture News* 38.3 (juin 2010), p. 60–71. ISSN : 0163-5964. DOI : 10.1145/1816038.1815971. URL : http://doi.acm.org/10.1145/1816038.1815971.

[37] Emmanuel JEANNOT et Guillaume MERCIER. « Improving MPI Applications Performance on Multicore Clusters with Rank Reordering ». In : *Recent Advances in the Message Passing Interface. The 18th European MPI User's Group Meeting (EuroMPI 2011)*. Lecture Notes in Computer Science. Santorini, Greece : Springer-Verlag, sept. 2011.

[38] Emmanuel JEANNOT et Guillaume MERCIER. « Near-optimal placement of MPI processes on hierarchical NUMA architecture ». In : *Proceedings of the 16th International Euro-Par Conference*. Lecture Notes in Computer Science. Ischia, Italy : Springer, août 2010.

[39] Emmanuel JEANNOT, Guillaume MERCIER et Francois TESSIER. « Process Placement in Multicore Clusters : Algorithmic Issues and Practical Techniques ». In : *IEEE Transactions on Parallel and Distributed Systems* 25.4 (avr. 2014), p. 993–1002. ISSN : 1045-9219. DOI : 10.1109/TPDS.2013.104.

[40] Hyun-Wook JIN et al. « Lightweight Kernel-Level Primitives for High-Performance MPI Intra-Node Communication over Multi-Core Systems ». In : *Proceedings of the IEEE International Conference on Cluster Computing (Cluster'07)*. Austin, TX, sept. 2007.

[41] Hyun-Wook JIN et al. « LiMIC : Support for High-Performance MPI Intra-Node Communication on Linux Cluster ». In : *Proceedings of the IEEE International Conference on Parallel Processing (ICPP-2005)*. Oslo, Norway : IEEE Computer Society Press, juin 2005.

[42] Sven KARLSSON et al. « MultiEdge : An Edge-based Communication Subsystem for Scalable Commodity Servers ». In : *Proceedings of the 21st International Parallel and Distributed Processing Symposium (IPDPS'07)*. Long Beach, CA, mar. 2007, p. 28.

[43] M. KOOP et al. « Performance Analysis and Evaluation of PCIe 2.0 and Quad-Data Rate InfiniBand ». In : *16th IEEE Int'l Symposium on Hot Interconnects (HotI16)*. Palo Alto, CA, août 2008.

[44] Ping LAI, Sayantan SUR et Dhabaleswar K. PANDA. « Designing truly one-sided MPI-2 RMA intra-node communication on multi-core systems ». In : *Proceedings of the International Supercomputing Conference (ISC'10)*. Hamburg, Germany, mai 2010.

[45] Christoph LAMETER. « NUMA (Non-Uniform Memory Access) : An Overview ». In : *ACM Queue* 11 (7 2013), p. 40.

[46] Jiuxing LIU et al. « Performance Comparison of MPI Implementations over InfiniBand, Myrinet and Quadrics ». In : *SuperComputing Conference (SC 03)*. Nov. 2003.

[47] Teng MA et al. « HierKNEM : An Adaptive Framework for Kernel-Assisted and Topology-Aware Collective Communications on Many-core Clusters ». In : *Proceedings of the 26th International Parallel and Distributed Processing Symposium (IPDPS'12)*. Shanghai, China, mai 2012.

[48] Teng MA et al. « Impact of Kernel-Assisted MPI Communication over Scientific Applications : CPMD and FFTW ». In : *Proceedings of the 18th European MPI Users Group Conference*. Lecture Notes in Computer Science. Santorini, Greece : Springer, sept. 2011.

[49] Teng MA et al. « Locality and Topology aware Intra-node Communication Among Multicore CPUs ». In : *Proceedings of the 17th European MPI Users Group Conference*. Lecture Notes in Computer Science. Stuttgart, Germany : Springer, sept. 2010.

[50] Milo M. K. MARTIN, Mark D. HILL et Daniel J. SORIN. « Why on-chip cache coherence is here to stay ». In : *Commun. ACM* 55.7 (juil. 2012), p. 78–89. ISSN : 0001-0782. DOI : 10.1145/2209249.2209269. URL : http://doi.acm.org/10.1145/2209249.2209269.

[51] Timothy G. MATTSON et al. « The 48-core SCC processor : the programmer's view ». In : *Proceedings of the 2010 ACM/IEEE conference on Supercomputing*. New Orleans, LA, nov. 2010.

[52] John D. MCCALPIN. « Memory Bandwidth and Machine Balance in Current High Performance Computers ». In : *IEEE Technical Committee on Computer Architecture (TCCA) Newsletter* (sept. 1995).

[53] Stéphanie MOREAUD. « Mouvement de données et placement des tâches pour les communications haute performance sur machines hiérarchiques ». Thèse de doct. 351 cours de la Libération — 33405 TALENCE cedex : Université Bordeaux 1, oct. 2011. URL : http://tel.archives-ouvertes.fr/tel-00635651.

[54] Scott PAKIN, Vijay KARAMCHETI et Andrew A. CHIEN. « Fast Messages (FM) : Efficient, Portable Communication for Workstation Clusters and Massively-Parallel Processors ». In : *IEEE Concurrency* 5 (1997), p. 60–73.

[55] Mark S. PAPAMARCOS et Janak H. PATEL. « A low-overhead coherence solution for multiprocessors with private cache memories ». In : *SIGARCH Comput. Archit. News* 12.3 (jan. 1984), p. 348–354. ISSN : 0163-5964. DOI : 10.1145/773453.808204. URL : http://doi.acm.org/10.1145/773453.808204.

[56] Stavros PASSAS, Kostas MAGOUTIS et Angelos BILAS. « Towards 100 Gbit/s Ethernet : Multicore-based Parallel Communication Protocol Design ». In : *Proceedings of the 23rd international conference on Supercomputing (ICS'09)*. ACM/SIGARCH. Yorktown Heights, NY, juin 2009, p. 214–224.

[57] Fabrizio PETRINI et al. « The Quadrics Network : High-Performance Clustering Technology ». In : *IEEE Micro* 22.1 (2002), p. 46–57.

[58] Bertrand PUTIGNY. « Benchmark-driven Approaches to Performance Modeling of Multi-Core Architectures ». Thèse de doct. 351 cours de la Libération — 33405 TALENCE cedex : Université Bordeaux, mar. 2014. URL : http://tel.archives-ouvertes.fr/tel-00984791.

[59] Mohammad J. RASHTI et Ahmad AFSAHI. « 10-Gigabit iWARP Ethernet : Comparative Performance Analysis with InfiniBand and Myrinet-10G ». In : *Proceedings of the International Workshop on Communication Architecture for Clusters (CAC), held in conjunction with IPDPS'07*. Long Beach, CA, mar. 2007, p. 234.

[60] Piyush SHIVAM, Pete WYCKOFF et D. K. PANDA. « EMP : Zero-copy OS-bypass NIC-driven Gigabit Ethernet Message Passing ». In : *Proceeding of Supercomputing ACM/IEEE 2001 Conference*. Denver, CO, nov. 2001, p. 57.

[61] Jeffrey M. SQUYRES et Andrew LUMSDAINE. « The Component Architecture of Open MPI : Enabling Third-Party Collective Algorithms ». In : *Proceedings, 18th ACM International Conference on Supercomputing, Workshop on Component Models and Systems for Grid Applications*. Sous la dir. de Vladimir GETOV et Thilo KIELMANN. St. Malo, France : Springer, juil. 2004, p. 167–185.

[62] Martin STECKERMEIER et Frank BELLOSA. *Using Locality Information in Userlevel Scheduling*. Rapp. tech. TR-95-14. Martensstraße 1, 91058 Erlangen, Germany : University of Erlangen-Nürnberg – Computer Science Department – Operating Systems – IMMD IV, déc. 1995.

[63] T. STERLING et al. « BEOWULF : A Parallel Workstation for Scientific Computation ». In : *Proceedings of the 24th International Conference on Parallel Processing*. Oconomowoc, WI, 1995, I :11–14.

[64] Shinji SUMIMOTO et Kouichi KUMON. « PM/Ehernet-kRMA : A High Performance Remote Memory Access Facility Using Multiple Gigabit Ethernet Cards ». In : *3rd International Symposium on Cluster Computing and the Grid (CCGrid2003)*. IEEE. Mai 2003, p. 326–334.

[65] H. TEZUKA et al. « Pin-down Cache : A Virtual Memory Management Technique for Zero-copy Communication ». In : *Proceedings of the 12th International Parallel Processing Symposium*. Avr. 1998, p. 308–315.

[66] Rajeev THAKUR et William GROPP. « Improving the Performance of Collective Operations in MPICH ». In : t. 2840. Lecture Notes in Computer Science. Springer, sept. 2003, p. 257–267.

[67] *The ZeptoOS project*. URL : http://www.mcs.anl.gov/research/projects/zeptoos/.

[68] Samuel THIBAULT. « Ordonnancement de processus légers sur architectures multiprocesseurs hiérarchiques : BubbleSched, une approche exploitant la structure du parallélisme des applications ». Thèse de doct. 351 cours de la Libération — 33405 TALENCE cedex : Université Bordeaux 1, déc. 2007. URL : http://tel.archives-ouvertes.fr/tel-00322881.

[69] Samuel THIBAULT, Raymond NAMYST et Pierre-André WACRENIER. « Building Portable Thread Schedulers for Hierarchical Multiprocessors : the BubbleSched Framework ». In : *EuroPar*. ACM. Rennes, France, août 2007. URL : http://hal.inria.fr/inria-00154506.

[70] *Top500 Supercomputing Sites*. http://top500.org.

[71] Jesper Larsson TRÄFF. « Implementing the MPI Process Topology Mechanism ». In : *Proceedings of the 2002 ACM/IEEE Conference on Supercomputing*. SC '02. Baltimore, Maryland : IEEE Computer Society Press, 2002.

[72] Leslie G. VALIANT. « A Bridging Model for Multi-core Computing ». In : *Proceedings of the 16th Annual European Symposium on Algorithms*. ESA '08. Karlsruhe, Germany : Springer-Verlag, 2008, p. 13–28. ISBN : 978-3-540-87743-1. DOI : 10.1007/978-3-540-87744-8_2. URL : http://dx.doi.org/10.1007/978-3-540-87744-8_2.

[73] Manjunath Gorentla VENKATA et al. « Open MPI for Cray XE/XK systems ». In : *Proceedings of the 2012 Cray User Group, Greengineering the Future*. Stuttgart, Germany, avr. 2012.

[74] Z. G. VRANESIC et al. « Hector - A Hierarchically Structured Shared Memory Multiprocessor ». In : *IEEE Computer* 24 (1991), p. 72–79.

[75] Samuel WILLIAMS et al. « The potential of the cell processor for scientific computing ». In : *Proceedings of the 3rd conference on Computing frontiers*. CF '06. Ischia, Italy : ACM, 2006, p. 9–20. ISBN : 1-59593-302-6. DOI : 10.1145/1128022.1128027. URL : http://doi.acm.org/10.1145/1128022.1128027.

[76] Wm. A. WULF et Sally A. MCKEE. « Hitting the memory wall : implications of the obvious ». In : *SIGARCH Computer Architecture News* 23.1 (mar. 1995), p. 20–24. ISSN : 0163-5964. DOI : 10.1145/216585.216588.

[77] Zhu YI et Peter P. WASKIEWICZ. « Enabling Linux Network Support of Hardware Multiqueue Devices ». In : *Proceedings of the Linux Symposium (OLS2007)*. Ottawa, Canada, juin 2007, p. 305–310.

Annexes

Annexe A
Curriculum Vitae

Résumé des sections suivantes :

Carrière et diplômes *Section A.1*

— Chargé de recherche Inria (depuis octobre 2006).
— Post-doctorat chez Myricom, Inc. (2005-2006).
— Thèse au Laboratoire de l'Informatique du Parallélisme, École normale supérieure de Lyon (2002-2005).
— Magistère et DEA d'informatique à l'ÉNS Lyon (1999-2002).

Recherche *Section A.2*

— LaBRI, équipe-projet Inria Runtime (depuis 2006).
— LIP (thèse), équipe-projet Inria RESO (2002-2005).
— **Thèmes de recherche** : programmation des réseaux rapides (thèse, post-doctorat, LaBRI), communication MPI intra-nœud (post-doctorat, LaBRI), MPI sur réseaux traditionnels (LaBRI), architectures NUMA (LaBRI), topologie des architectures (LaBRI).
— **Collaborations internationales** : Argonne National Laboratory, Oak Ridge National Laboratory, Université du Tennessee Knoxville, Université de San Luis (Argentine), Université de la République (Uruguay), Myricom, Cisco.
— **Publications** : 4 revues internationales, 15 conférences internationales, 11 workshops internationaux, 2 livres.

Activités pédagogiques *Section A.4*

— Cours, TDs et projets de Système d'exploitation et Systèmes parallèles distribués, 2ème et 3ème années de l'ENSEIRB-MATMECA (Talence, depuis 2006).
— TDs de Système d'exploitation et Programmation système en Licence et Master à l'ÉNS Lyon et l'Université de Lyon (2002-2005).

Médiation scientifique *Section A.4*

— Chargé de médiation scientifique d'Inria Bordeaux - Sud-Ouest (depuis 2010).
— 4 articles Interstices, Fête de la Science, Salons de recrutement, etc.

Responsabilités administratives *Section A.5*

— École d'été CEA-EDF-Inria sur la programmation des architectures hétérogènes (2013).
— Stand Inria à SuperComputing (depuis 2010).
— Représentant d'Inria dans le consortium Open MPI et le MPI Forum (depuis 2010).
— Représentant du personnel dans Comité de centre et CLHSCT (depuis 2008).
— Marchés publics pour achat de calculateurs (depuis 2009).
— Jurys de concours externe pour service support Inria (depuis 2011).
— Administration de la plate-forme expérimentale de l'équipe (depuis 2007).

Liste complète des publications *Section A.6*

A.1 Curriculum vitae

A.1.1 Informations personnelles

Brice Goglin
Né en 1978, à Reims (51).
Marié, 1 enfant.

Adresse : 7 parc des Charmettes – 33700 Merignac.
Téléphone : 09 54 10 78 90 – 06 84 85 99 12.

Adresse électronique : Brice.Goglin@inria.fr
Page web professionnelle : http://runtime.bordeaux.inria.fr/goglin

A.1.2 Carrière

depuis 2008 Chargé de recherche Inria de première classe, équipe Satanas du LaBRI, équipe-projet Runtime d'Inria Bordeaux Sud-Ouest.

2006-2008 Chargé de recherche Inria de seconde classe, équipe Satanas du LaBRI, équipe-projet Runtime d'Inria Bordeaux Sud-Ouest.

2005-2006 Séjour post-doctoral dans l'entreprise Myricom (Oak Ridge, Tennessee)

A.1.3 Diplômes

2002-2005 Doctorat d'Informatique (École normale supérieure de Lyon, Laboratoire d'Informatique du Parallélisme), sous la direction de Pascale Vicat-Blanc Primet et Olivier Glück (initiée sous la direction de Loïc Prylli),
Réseaux rapides et stockage distribué dans les grappes de calculateurs : propositions pour une interaction efficace,
soutenue devant un jury composé de Thierry Priol (président du jury), Raymond Namyst et Brigitte Plateau (rapporteurs), Yves Denneulin et Bernard Lecussan.

1999-2002 Magistère d'Informatique et Modélisation (ÉNS Lyon) : Licence et Maîtrise d'Informatique, puis DEA d'Informatique Fondamentale.

1998-2000 Licence et Maîtrise de Physique (ÉNS Lyon).

1996-1998 Classes préparatoires aux grandes écoles MPSI et MP* (Lycée Clemenceau, Reims), reçu au concours Informatique de l'ÉNS Lyon.

A.2 Recherche

Mes activités de recherche se situent dans le domaine du calcul haute performance (HPC), dans les couches logicielles basses permettant leur exploitation efficace, en particulier le système d'exploitation et les supports exécutifs.

Thèse Après des stages de maîtrise et DEA consacrés à l'étude de l'adaptabilité des réseaux rapides du HPC à des besoins différents des communications MPI habituelles, ma thèse [Th1, RF1, RR3] m'a conduit à proposer et à implémenter des améliorations des interfaces de programmation de ces réseaux afin de faciliter leur utilisation pour le stockage distribué.

Après avoir développé un prototype d'expérimentation dédié [WI6, CI11], j'ai montré comment résoudre les problèmes d'interaction entre l'interface de programmation GM des réseaux Myrinet et la pile d'accès au stockage distant [WI7]. J'ai ensuite intégré ces travaux dans la nouvelle interface de programmation MX afin qu'elle réponde nativement à ces besoins [CI10, CF1].

Post-doctorat La collaboration mise en place pendant cette thèse m'a permis d'effectuer ensuite mon post-doctorat chez Myricom, où j'ai poursuivi mes travaux sur les interfaces de programmation des réseaux rapides.

Mes résultats de thèse ont été validés à travers le portage des systèmes de stockage distribué Lustre et PVFS sur MX [RR1]. J'ai également revisité les vieilles innovations (zéro-copie et OS-bypass) dans le contexte des plates-formes modernes [RR2] et mis en évidence l'inadéquation entre le noyau Linux et le HPC [RA1] (résultats repris dans ma conclusion 6.3.3).

LaBRI et Inria Bordeaux - Sud-Ouest Lors de mon arrivée dans l'équipe-projet Inria Runtime à Bordeaux en 2006, j'ai naturellement continué mes recherches sur les réseaux rapides. Mais la démocratisation des processeurs multicœurs et des architectures NUMA dans les grappes de calcul m'a amené à étudier leur impact sur les communications réseau, et de façon plus générale sur la façon de maîtriser la complexité croissance des plates-formes de calcul parallèle. Je présente ci-dessous les 5 principaux axes de recherche que j'ai développés depuis mon arrivée à Bordeaux.

Passage de messages sur réseaux Ethernet (chapitre 2)

J'ai étudié l'adaptation du passage de messages (communications MPI pour le calcul haute performance) sur des réseaux traditionnels (Ethernet) car ces technologiques représentent la plupart des grappes de calcul (la moitié du Top500). Ce travail s'est inscrit dans une collaboration contractuelle avec Myricom qui visait à déployer leur protocole MX sur des plates-formes ne disposant pas de leur technologie réseau.

J'ai montré qu'il était possible d'obtenir des performances satisfaisantes dans ce contexte sans imposer de modifier le matériel ni les couches réseau logicielles comme les implémentations concurrentes le font. Ce modèle m'a ensuite permis de proposer des optimisations originales comme le recouvrement de copies mémoire déportées du coté récepteur, pour les communications inter- et intra-nœuds. Et j'ai montré que de petites modifications du matériel Ethernet permettaient au passage de messages de bénéficier d'optimisations similaires à TCP concernant les interruptions et la gestions des affinités en réception.

Ce travail a été mené dans le logiciel Open-MX en collaboration avec Myricom et est désormais réévalué dans CCI en collaboration avec Oak Ridge National Lab. Il a donné lieu à 8 publications [RI3], [RI2], [CI7], [CI8], [CI4], [CI5], [WI2], [WI3].

Communication MPI intra-nœud (chapitre 3)

J'ai généralisé certaines optimisations proposées dans l'axe de recherche précédent en proposant des communications MPI intra-nœuds par copie directe entre processus indépendamment du matériel réseau. Nous avons ensuite montré qu'il était nécessaire de tenir compte de la topologie matérielle (en particulier les caches) pour décider dynamiquement quelle stratégie de communication utiliser. Nous avons enfin proposé un modèle permettant de simplifier significativement l'implémentation d'opérations collectives hiérarchiques et pipelinées.

Ce travail a été mené à travers le logiciel KNEM, dans le cadre de la thèse de Stéphanie Moreaud, en collaboration avec Argonne National Lab (dans le cadre de l'équipe associée MPI-Runtime), puis avec l'Université du Tennessee Knoxville (Innovative Computing Lab) et Cisco. Il a donné lieu à 5 publications [RI4], [CI12], [CI4], [CI3], [WI8].

Modélisation des architectures hiérarchiques (chapitre 4)

Nos travaux sur les architectures NUMA et NUIOA ont imposé une connaissance fine du matériel. Nous avons proposé une modélisation abstraite et portable de la topologie matérielle sous la forme d'arbres de ressources permettant d'exposer les affinités entre les différents composants, cœurs, caches, mémoires... Ce travail permet aux bibliothèques parallèles de placer les tâches selon leurs affinités, par exemple en plongeant un arbre de processus dans notre arbre décrivant le matériel.

Ce travail est mené dans le logiciel hwloc, en collaboration avec le consortium Open MPI et Cisco, et dans le cadre de la thèse de Bertrand Putigny. Il a donné lieu à 4 publications [CI1], [WI9], [RA2], [CI6], [CI15], [WI10].

Par ailleurs, ce travail est en cours d'extension à la topologie des réseaux en collaboration avec Cisco et l'Université du Wisconsin - La Crosse, dans le cadre du consortium Open MPI, autour du logiciel netloc.

Migration mémoire dans les architectures NUMA (section 5.2)

Dans le cadre de la thèse de François Broquedis qui étudiait l'ordonnancement hiérarchique de threads OpenMP sur architectures NUMA, j'ai proposé des stratégies de migration mémoire optimisées. Ce travail a permis de mettre en place des stratégies de placement conjoint des threads OpenMP et de la mémoire très utiles pour les applications parallèles irrégulières, en particulier si les affinités entre tâches et données varient au cours du temps.

Une partie des modifications de fonctionnalités de migration mémoire a été intégrée dans le noyau Linux officiel. Cet axe de recherche a donné lieu à 6 publications [RI1], [CI2], [CI9], [WI4], [WI1]. [WI11].

Communications dans les architectures NUIOA (section 5.3)

J'ai également étudié l'impact des architectures NUMA sur les communications réseau. Cet aspect, à la croisée des domaines historiques des grappes de calcul et des supercalculateurs à mémoire partagée, devient incontournable suite à la généralisation des architectures multicœurs et NUMA.

Nous avons été pionniers de cet axe de recherche, introduisant l'acronyme NUIOA pour *Non Uniform Input/Output Access*, et l'avons décliné sous deux aspects. D'une part, nous avons montré l'importance de placer les tâches selon leurs affinités pour le matériel réseau. D'autre part, une fois le placement effectué, nous avons mis en évidence la nécessité d'adapter les stratégies de communication, en répartissant les données entre les différentes cartes réseau selon leur localité, et en choisissant les processus intermédiaires près du matériel réseau.

Ce travail a été mené dans le cadre de la thèse de Stéphanie Moreaud, en collaboration avec le consortium Open MPI, et a donné lieu à 3 publications [CI14], [CI13], [WI5].

A.2.1 Collaborations

Voici la liste des collaborations que j'ai mises en place ou auxquelles j'ai participé, avec les publications en commun.

Collaborations mises en place

STIC-AmSud Responsable du projet collaboratif *SEHLOC* (paramétrisation de la topologie matérielle dans le modèle Multi-BSP) avec UNSL (San Luis, Argentine) et UdelaR (Uruguay) (32 k€, 2013-2014). `http://runtime.bordeaux.inria.fr/sehloc`.
- — Séjour en Uruguay : 10 jours (avril 2013).
- — Séjour en Argentine : 1 semaine (décembre 2013).

Open MPI consortium Intégration du logiciel hwloc [RA2] dans Open MPI, puis développement des fonctionnalités de gestion des affinités matérielles dans Open MPI, puis développement de netloc (depuis 2009).

Myricom, Inc. Collaboration initiée pendant ma thèse autour du développement de l'interface de programmation MX (2004-2005), poursuivie à travers mon post-doctorat chez eux (2005-2006), puis renouvelée autour du développement d'Open-MX dans un cadre contractuel (37 k€ et don de matériel, 2008-2009).
- — Séjours en Californie : 3 mois (stage de maîtrise, été 2001), 3 semaines (thèse, janvier 2004), 1 semaine (Open-MX, mai 2008).
- — Séjours dans le Tennessee : 1 mois (thèse, été 2004), 11 mois (post-doctorat, de octobre 2005 à septembre 2006).

Participation à des collaborations

Oak Ridge National Laboratory Développement du logiciel CCI (depuis 2011).

Argonne National Laboratory Équipe associée Inria *MPI-Runtime* (2009-2011, gérée par Guillaume Mercier), développement du logiciel KNEM [CI3], [WI8] puis intégration de hwloc dans l'implémentation MPICH.
- — Séjour : 1 semaine (décembre 2008).

Projets ANR Participation à
- — SONGS (*Simulation Of Next Generation Systems*), depuis 2012 ;
- — PARA (parallélisme et productivité des applications), 2007-2009 ;
- — NUMASIS (évaluation et optimisation d'applications sismiques sur architectures NUMA), 2007-2009.

Collaborations informelles

Université du Wisconsin-La Crosse et Cisco Extension du logiciel hwloc aux réseaux de serveurs, logiciel netloc (depuis 2013).

Bull (Echirolles) Utilisation des logiciels hwloc et KNEM dans les outils Bull (2012).

Université du Tennessee Knoxville et Cisco Intégration du support logiciel pour le pilote noyau KNEM dans Open MPI [CI12] (2010-2011).

A.2.2 Logiciels

Open-MX

Open-MX est une pile de passage de messages proposant l'interface des réseaux Myricom au dessus de n'importe quel matériel Ethernet. J'en suis le créateur, dans le cadre d'une collaboration avec Myricom, et principal développeur.
État : Stable, activités de recherche et développement terminées.
Taille : 45k lignes de C (95% par moi-même), dont 13k dans le noyau Linux.
Distribution : Licence LGPL, depuis http://open-mx.gforge.inria.fr.
Dépôt APP : IDDN.FR.001.290034.001.S.P.2008.000.10100.

CCI

CCI (*Common Communication Interface*) est le successeur et une généralisation d'Open-MX a des domaines plus vastes que le simple passage de messages MPI pour le HPC. Il est conçu en collaboration avec principalement Oak Ridge National Lab.
État : Beta, développement.
Taille : 40k lignes de C (25% par moi-même), dont 4k dans le noyau Linux.
Distribution : Licence BSD, depuis http://cci-forum.org.

KNEM

KNEM (*Kernel Nemesis*) est un module pour le noyau Linux offrant des communications MPI intra-nœuds optimisées. Il a été conçu en collaboration avec Argonne National Lab puis étendu avec l'Université du Tennessee Knoxville et Cisco.
État : Stable, activité de recherche terminée.
Taille : 8k lignes de C (100% par moi-même), dont 4k dans le noyau Linux.
Distribution : Licence BSD, depuis http://runtime.bordeaux.inria.fr/knem.
Dépôt APP : IDDN.FR.001.510027.000.S.P.2010.000.31235.

J'ai également contribué au support de KNEM dans les suites MPICH et Open MPI.

hwloc

hwloc (*Hardware Locality*) est une bibliothèque de détection de topologie matérielle (intérieur des serveurs) et de placement des tâches et données. Elle est très utilisée, et est notamment embarquée dans toutes les implémentations MPI populaires, de nombreux gestionnaires de batchs, certaines bibliothèques parallèles et quelques compilateurs. hwloc attire des contributions de nombreux collaborateurs, académiques et industriels.
État : Stable, mais toujours en développement.
Taille : 30k lignes de C (60% par moi-même).
Distribution : Licence BSD, depuis http://www.open-mpi.org/projects/hwloc/.

netloc

netloc (*Network Locality*) est le compagnon de hwloc pour la détection des topologies des réseaux, en particulier Ethernet et InfiniBand. Il est en développement depuis 2013 en collaboration avec l'Université du Wisconsin La Crosse et Cisco.
État : Beta, en développement.
Taille : 13k de lignes de C, 2k lignes de Perl (50% par moi-même).
Distribution : Licence BSD, depuis http://www.open-mpi.org/projects/netloc/.

Noyau Linux

Modifications les plus significatives que j'ai intégrées au noyau Linux officiel (parmi près de 200 patchs en tout) :

- Répartition des canaux DMA I/OAT selon les affinités PCI-NUMA (Linux 3.12) ;
- Réactivation des copies DMA I/OAT non alignées quand les opérations RAID sont mal supportées (3.12) ;
- Récupération des compteurs de performances `perf` par thread (2.6.32) ;
- Amélioration des performances de migration mémoire (entre 2.6.28 et 2.6.31) ;
- Exposition de la localité PCI-NUMA dans le système de fichiers virtuels `sysfs` (2.6.21) ;
- Amélioration du support des *Message Signaled Interrupts* pour activation par défaut malgré les bugs matériels (2.6.18 et 2.6.19) ;
- Intégration puis maintenance du pilote `myri10ge` des cartes Ethernet 10G de Myricom (plus de 130 patchs entre 2.6.18 et 2.6.34).

A.2.3 Encadrement

Jeunes chercheurs

Doctorat :

— Bertrand Putigny (2010-2014, co-encadrement avec Denis Barthou, [58]), désormais ingénieur de recherche à l'Institut de Recherche en Astrophysique et Planétologie (IRAP, Toulouse).

— Stéphanie Moreaud (2007-2011, co-encadrement avec Raymond Namyst, [53]), désormais en post-doctorat à l'Université du Tennessee Knoxville.

Stage M2 Recherche :

— Nicolas Denoyelle (2014).

— François Broquedis (2007), désormais maître de conférences à l'ENSIMAG (Grenoble).

— Stéphanie Moreaud (2007).

Étudiants

Stages étudiants :

— Benoît Ruelle (2ème année Informatique, ENSEIRB-MATMECA, 2013), désormais ingénieur chez OVH (France).

— Antoine Rougier (Master 1, Université de Bordeaux, 2012), désormais doctorant à l'Université de Vienne (Autriche).

— Romain Perier (Master 1, Université de Bordeaux, 2010), désormais ingénieur expert au LaBRI (Bordeaux).

Projets étudiants : Projet M2 professionnel (2010) et Projet de Programmation M1 (2012), Université Bordeaux 1.

Tutorat de projet de fin d'étude : 7 étudiants depuis 2010 (ENSEIRB-MATMECA).

Initiation à la recherche : 6 étudiants encadrés (2009 et 2010, M2 Recherche, Université Bordeaux 1).

Ingénieurs

Ingénieur Jeune Diplômé :

— Ludovic Stordeur (2009-2011, action de développement technologique autour du logiciel Open-MX), désormais ingénieur dans l'entreprise QOSMOS (Paris).

A.2.4 Les cinq publications les plus significatives

[CI5] Brice GOGLIN. « Improving Message Passing over Ethernet with I/OAT Copy Offload in Open-MX ». In : *Proceedings of the IEEE International Conference on Cluster Computing*. Tsukuba, Japan : IEEE Computer Society Press, sept. 2008, p. 223–231. DOI : 10.1109/CLUSTR.2008.4663775. URL : http://hal.inria.fr/inria-00288757

[RI4] Brice GOGLIN et Stéphanie MOREAUD. « KNEM : a Generic and Scalable Kernel-Assisted Intra-node MPI Communication Framework ». In : *Journal of Parallel and Distributed Computing (JPDC)* 73.2 (fév. 2013), p. 176–188. DOI : 10.1016/j.jpdc.2012.09.016. URL : http://hal.inria.fr/hal-00731714

[CI1] François BROQUEDIS et al. « hwloc : a Generic Framework for Managing Hardware Affinities in HPC Applications ». In : *Proceedings of the 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP2010)*. Pisa, Italia : IEEE Computer Society Press, fév. 2010, p. 180–186. DOI : 10.1109/PDP.2010.67. URL : http://hal.inria.fr/inria-00429889

[RI1] François BROQUEDIS et al. « ForestGOMP : an efficient OpenMP environment for NUMA architectures ». In : *International Journal on Parallel Programming, Special Issue on OpenMP ; Guest Editors : Matthias S. Müller and Eduard Ayguadé* 38.5 (2010), p. 418–439. DOI : 10.1007/s10766-010-0136-3. URL : http://hal.inria.fr/inria-00496295

[CI14] Stéphanie MOREAUD, Brice GOGLIN et Raymond NAMYST. « Adaptive MPI Multi-rail Tuning for Non-Uniform Input/Output Access ». In : *Recent Advances in the Message Passing Interface. The 17th European MPI User's Group Meeting (EuroMPI 2010)*. Sous la dir. d'Edgar Gabriel RAINER KELLER et Jack DONGARRA. T. 6305. Lecture Notes in Computer Science. Best paper award. Stuttgart, Germany : Springer-Verlag, sept. 2010, p. 239–248. DOI : 10.1007/978-3-642-15646-5_25. URL : http://hal.inria.fr/inria-00486178

Ces cinq publications sont détaillées à la fin de ce manuscrit (chapitre B). La liste complète des publications est disponible en section A.6.

A.2.5 Prix

Best paper award à EuroMPI 2010 [CI14].

A.2.6 Présentations invitées

- Open MPI *State of the Union* à SuperComputing : *hwloc (Hardware Locality)* (Nouvelle Orléans, 2010/11) et *netloc (Network Locality)* (Denver, 2013/11).
- Stand Cisco à SuperComputing : *High-Performance Message Passing over generic Ethernet with Open-MX* (Portland, 2009/11) et *Hardware Locality (hwloc) : Do you really know where your processes are running ?* (Nouvelle Orléans, 2010/11).
- Journées grilles de calcul : *Grid'5000 : une plate-forme d'expérimentation pour les systèmes distribués à large échelle* (Bordeaux, 2010/06).
- Interview Podcast RCE : *Open-MX* (http://www.rce-cast.com/, 2009).
- STFC Daresbury Laboratory : *High-Performance Message Passing over Ethernet with Open-MX... Why and How ?* (Angleterre, 2009/01).
- Clustervision Users Group : *The Borderline Cluster : Experimenting with Bleeding-Edge Interconnects* (Cheltenham, Angleterre, 2007/10).

A.2.7 Autres présentations

- Semaine Sports-Études du Département d'Informatique de l'ÉNS Lyon : *Localité matérielle et affinités logicielles dans le calcul haute-performance* (Le Pleynet, 2014/01).
- Rencontre Inria Industrie : *Mieux placer les processus pour améliorer les performances des applications parallèles* (Paris, 2013/06).
- Joint Lab for Petascale Computing : *Bringing hardware affinity information into MPI communication strategies* (Rennes, 2012/06).
- iMatch HPC : *hwloc : Portable Hardware Locality* (Bordeaux, 2012/06).
- Rencontres Mondiales du Logiciel Libre : *Kernel driver maintenance : Upstream vs. Industry* et *Linux vs HPC : Life (and death) of strange features* (Bordeaux, 2011/07).
- Bull : *Entrées-sorties dans les grappes Myrinet : analyse au travers du portage de Lustre sur MX* (Echirolles, 2006/04).
- Sun Microsystems : *ORFA, Optimizing Access to Remote File Systems* (Grenoble, 2003/07).

A.3 Enseignements

A.3.1 Cours à l'Université, à l'ÉNS Lyon et en école d'ingénieur

depuis 2008 Cours, TDs et projets de Système d'exploitation en 2ème année d'Informatique de l'ENSEIRB-MATMECA. 140h de cours magistraux et environ 130h de TD et projet en tout. Responsable de l'unité d'enseignement correspondante.

2012 Validation des Acquis de l'Expérience (VAE) de Daniel Mouly (ENSEIRB-MATMECA). Rapporteur et membre du jury.

2006-2010 Création du cours de Systèmes Parallèles et Distribués en 3ème année d'Informatique de l'ENSEIRB-MATMECA. Environ 40h de cours magistraux et 15h de TP en tout.

2002-2005 Monitorat à l'École normale supérieure de Lyon, et les Université Lyon 1 et 2. 192h éq. TD (dont 15h de cours magistraux) : initiation à l'informatique, programmation système et système d'exploitation.

A.3.2 Formation aux chercheurs et ingénieurs

- Tutoriel sur le logiciel hwloc
  — ComPAS 2014, Neufchâtel (Suisse), le 22 avril 2014 ;
  — ComPAS 2013, Grenoble, 15 janvier 2013 ;
  — HiPEAC 2013, Berlin, 21 janvier 2013.
- Séminaires *les mardis du développement technologique* (Service Expérimentation et Développement d'Inria Bordeaux Sud-Ouest)
  — GIT (2011/05) ;
  — Faire des releases logicielles : Pourquoi, Quand et Comment ? (2013/02).

A.4 Médiation scientifique

A.4.1 Aspects organisationnels

- Chargé de médiation scientifique d'Inria Bordeaux - Sud-Ouest (depuis 2010), responsable du groupe de travail de médiation scientifique du centre, en collaboration avec le service communication.
- Organisation des Ateliers médiation de l'Inria Bordeaux - Sud-Ouest et du master de Médiation de Sciences de Bordeaux 3 (2012).
- Accueil d'élèves pour des ateliers et présentation dans le centre (Fête de la Science : environ 300 collégiens ou lycéens par an ; Terminale ISN : environ 55 élèves en 2013/02 ; Stage MathsC2+ : environ 35 élèves de seconde en 2014/04 ; École des Ponts et chaussées : environ 25 élèves en 2012/09).
- Organisation de 4 sessions de formation en médiation scientifique pour 30 chercheurs d'Inria Bordeaux - Sud-Ouest (depuis 2011).
- Participation aux ateliers diffusion des Sciences et Techniques du conseil régional, dans le cadre du schéma régional de l'enseignement supérieur, de la recherche et de l'innovation (SRESRI, 2012).
- Participation aux tables rondes d'organisation de la Semaine Digitale de la ville de Bordeaux, sur le sujet de la promotion des filles dans les carrières numériques (2013).
- Coordinateur de la session Systèmes d'exploitation des Rencontres Mondiales du Logiciel Libre (Talence, 2010).

A.4.2 Interventions

- Fête de la Science (tous les ans depuis 2010) ;
- Présentation des métiers de la recherche sur le stand Inria au salon Aquitec (tous les ans depuis 2010), conférence de présentation des métiers de la science (2010 et 2012) ;
- Ateliers métiers ENSEIRB-MATMECA (2010) ;
- Nuit des chercheurs (2010) ;
- Banquet des scientifiques de CapSciences (2010) ;
- 2 présentations aux Rencontres Mondiales du Logiciel Libre (Talence, 2010).

A.4.3 Autres

- Accueil de stagiaires de 3ème (stage de découverte professionnelle, 2012) ;
- Interview TechnoScope/Inria lors du salon SuperComputing 2012 ;
- Interviews France Bleu Gironde et Radio Campus lors de l'inauguration du nouveau bâtiment Inria (2012) ;
- Articles de médiation scientifique (Interstices, 2011 [MS1], [MS2], et 2013 [MS4], [MS3]) ;
- Marque page *Visages des sciences* de présentation du métier de chercheur (2009).

A.5 Activités administratives et responsabilités collectives

A.5.1 Au sein d'Inria

- Coordinateur scientifique de l'école d'été CEA-EDF-Inria 2013 sur la programmation des architectures hétérogènes :
  — 30h de cours par Michael Wolfe (the Portland Group), Josef Weidendorfer (Technische Universität München), Wen-Mei Hwu (University of Illinois at Urbana Champaign).
  — 20h de travaux pratiques.
  — 4h de présentations industrielles (Intel, NVIDIA, CAPS Entreprise).
  — 21 participants, 7 nationalités.
- Membre du comité national de médiation scientifique (depuis 2012) qui réfléchit au plan de communication scientifique de l'institut.
- Organisateur scientifique et logistique du stand Inria à SuperComputing en 2011 et 2012, co-organisateur en 2013, coordinateur scientifique en 2010.
- Membre des groupes de travail :
  — Plates-formes de portage logiciel (direction du développement technologique, 2012-2013) ;
  — Évaluation des activités de médiation scientifique (commission d'évaluation, 2011-2012) ;
  — Médiation scientifique (direction de la communication, 2011).
- Représentant d'Inria au sein du consortium Open MPI (développement logiciel).

A.5.2 Au sein d'Inria Bordeaux - Sud-Ouest

- Chargé de médiation scientifique (depuis 2010).
- Représentant du personnel au sein du comité de centre (depuis 2008).
- Représentant du personnel au sein du comité local d'hygiène et sécurité et des conditions de travail (2009-2012).
- Organisateur de la journée PlaFRIM en avril 2013 pour partager les usages de la plate-forme de calcul parallèle de l'Inria Bordeaux - Sud-Ouest, du LaBRI et de l'IMB.

A.5.3 Au sein de l'équipe

- Représentant de l'équipe Satanas pour le patrimoine et la sécurité des systèmes d'information (LaBRI, depuis 2007).
- Administration de la plate-forme expérimentale de l'équipe (15 serveurs, 40 utilisateurs) et gestion du matériel informatique.
- Représentant au sein du comité utilisateur de la plate-forme expérimentale PlaFRIM (LaBRI/IMB/Inria, depuis 2009).

A.5.4 Comités de programme

J'ai fait partie des comités de programme des conférences internationales suivantes :

- SuperComputing 2012 (technical program committee, technical poster committee, ACM student research competition jury) ;
- HIPC 2013 et 2014 ;
- Cluster 2014 ;
- EuroMPI 2011, 2012, 2013, et EuroMPI/ASIA 2014 ;

- ICCCN 2014 ;
- CARLA 2014 (joint HPC LatAm + CLCAR conference) ;
- Hot Interconnects 2012, 2013 et 2014 ;
- ISPAN 2011 ;
- *Local Chair* du topic *Réseaux* d'Euro-Par 2011 (organisé à Bordeaux).

J'ai de plus été chair de session lors des conférences internationales ICPP (2011) et Euro-Par (2011), et du workshop international P2S2 (2011).

J'ai par ailleurs reviewé des articles pour
- les journaux internationaux IEEE Network (2013), JPDC (2011) ;
- les conférences internationales Cluster (2007), EuroPVM/MPI (2007), CCGrid (2008-2009, 2011-2013), IPDPS (2010, 2013), PLDI (2011), HPDC (2012), PACT (2013), Euro-Par (2014) ;
- les workshops internationaux HPIDC (2009), CASS (2011), ROSS (2012).

A.5.5 Standardisation

- Représentant d'Inria au MPI Forum (en charge de la définition du standard MPI) depuis septembre 2010. Contribution aux groupes de travail *Remote Memory Access*, *Hybrid*, etc.).

A.5.6 Marchés publics

- Responsable du marché adapté pour l'achat des serveurs multi-GPU de l'équipe (Inria, 2012, 40 k€).
- Responsable du marché adapté PRI NUMA (LaBRI, 2009, 150 k€) pour l'achat d'un serveur NUMA à mémoire partagée (LaBRI).
- Membre des commissions des appels d'offre pour la plate-forme PlaFRIM (LaBRI, IMB, Inria) : 400 k€ pour 68 serveurs en 2009, 100 k€ pour 8 serveurs multi-GPU en 2010, 100 k€ pour un serveur NUMA en 2011, 300 k€ pour un cluster de Xeon Phi en 2013.

A.5.7 Jurys de concours

- Ingénieur expert en architecture système et réseau (Service Informatique du Centre, Inria Bordeaux - Sud-Ouest, 2013) ;
- Assistant de partenariats, projets et innovation (Service Transfert et Innovation, Inria Bordeaux - Sud-Ouest, 2012) ;
- Assistant de communication (Service Communication, Inria Bordeaux - Sud-Ouest, 2011).

A.5.8 Autres responsabilités

- Trésorier du comité local de l'AGOS (comité d'entreprise d'Inria) depuis 2012.
- Mainteneur du pilote *myri10ge* pour les cartes Ethernet 10G de Myricom dans le noyau Linux (2006-2010).
- Développeur Debian en charge des paquets du serveur graphique X.org (2006-2010).
- Second *Gate Keeper* de la branche stable 1.4 de Open MPI (chargé d'appliquer les correctifs après validation).

A.6 Liste des publications

Revues internationales avec comité de lecture

[RI1] François BROQUEDIS et al. « ForestGOMP : an efficient OpenMP environment for NUMA architectures ». In : *International Journal on Parallel Programming, Special Issue on OpenMP ; Guest Editors : Matthias S. Müller and Eduard Ayguadé* 38.5 (2010), p. 418–439. DOI : 10.1007/s10766-010-0136-3. URL : http://hal.inria.fr/inria-00496295.

[RI2] Brice GOGLIN. « High-Performance Message Passing over generic Ethernet Hardware with Open-MX ». In : *Elsevier Journal of Parallel Computing (PARCO)* 37.2 (fév. 2011), p. 85–100. DOI : 10.1016/j.parco.2010.11.001. URL : http://hal.inria.fr/inria-00533058.

[RI3] Brice GOGLIN. « NIC-assisted Cache-Efficient Receive Stack for Message Passing over Ethernet ». In : *Concurrency and Computation : Practice and Experience, Special Issue : Euro-Par 2009* 23 (2 fév. 2011), p. 199–210. DOI : 10.1002/cpe.1632. URL : http://hal.inria.fr/inria-00496301.

[RI4] Brice GOGLIN et Stéphanie MOREAUD. « KNEM : a Generic and Scalable Kernel-Assisted Intra-node MPI Communication Framework ». In : *Journal of Parallel and Distributed Computing (JPDC)* 73.2 (fév. 2013), p. 176–188. DOI : 10.1016/j.jpdc.2012.09.016. URL : http://hal.inria.fr/hal-00731714.

Conférences internationales avec comité de lecture

[CI1] François BROQUEDIS et al. « hwloc : a Generic Framework for Managing Hardware Affinities in HPC Applications ». In : *Proceedings of the 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP2010)*. Pisa, Italia : IEEE Computer Society Press, fév. 2010, p. 180–186. DOI : 10.1109/PDP.2010.67. URL : http://hal.inria.fr/inria-00429889.

[CI2] François BROQUEDIS et al. « Structuring the execution of OpenMP applications for multicore architectures ». In : *Proceedings of 24th IEEE International Parallel and Distributed Processing Symposium (IPDPS'10)*. Atlanta, GA : IEEE Computer Society Press, avr. 2010. DOI : 10.1109/IPDPS.2010.5470442. URL : http://hal.inria.fr/inria-00441472.

[CI3] Darius BUNTINAS et al. « Cache-Efficient, Intranode Large-Message MPI Communication with MPICH2-Nemesis ». In : *Proceedings of the 38th International Conference on Parallel Processing (ICPP-2009)*. Vienna, Austria : IEEE Computer Society Press, sept. 2009, p. 462–469. DOI : 10.1109/ICPP.2009.22. URL : http://hal.inria.fr/inria-00390064.

[CI4] Brice GOGLIN. « High Throughput Intra-Node MPI Communication with Open-MX ». In : *Proceedings of the 17th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP2009)*. Weimar, Germany : IEEE Computer Society Press, fév. 2009, p. 173–180. DOI : 10.1109/PDP.2009.20. URL : http://hal.inria.fr/inria-00331209.

[CI5] Brice GOGLIN. « Improving Message Passing over Ethernet with I/OAT Copy Offload in Open-MX ». In : *Proceedings of the IEEE International Conference on Cluster Computing*. Tsukuba, Japan : IEEE Computer Society Press, sept. 2008, p. 223–231. DOI : 10.1109/CLUSTR.2008.4663775. URL : http://hal.inria.fr/inria-00288757.

[CI6] Brice GOGLIN. « Managing the Topology of Heterogeneous Cluster Nodes with Hardware Locality (hwloc) ». In : *Proceedings of 2014 International Conference on High Performance Computing & Simulation (HPCS 2014)*. Bologna, Italy, juil. 2014. URL : http://hal.inria.fr/hal-00985096.

[CI7] Brice GOGLIN. « NIC-assisted Cache-Efficient Receive Stack for Message Passing over Ethernet ». In : *Proceedings of the 15th International Euro-Par Conference*. T. 5704. Lecture Notes in Computer Science. Delft, The Netherlands : Springer, août 2009, p. 1065–1077. DOI : 10.1007/978-3-642-03869-3_98. URL : http://hal.inria.fr/inria-00379168.

[CI8] Brice GOGLIN et Nathalie FURMENTO. « Finding a Tradeoff between Host Interrupt Load and MPI Latency over Ethernet ». In : *Proceedings of the IEEE International Conference on Cluster Computing*. New Orleans, LA : IEEE Computer Society Press, sept. 2009. DOI : 10.1109/CLUSTR.2009.5289165. URL : http://hal.inria.fr/inria-00397328.

[CI9] Brice GOGLIN et Nathalie FURMENTO. « Memory Migration on Next-Touch ». In : *Proceedings of the Linux Symposium*. Montreal, Canada, juil. 2009, p. 101–110. URL : http://hal.inria.fr/inria-00378580.

[CI10] Brice GOGLIN, Olivier GLÜCK et Pascale Vicat-Blanc PRIMET. « An Efficient Network API for in-Kernel Applications in Clusters ». In : *Proceedings of the IEEE International Conference on Cluster Computing*. Boston, Massachussets : IEEE Computer Society Press, sept. 2005. DOI : 10.1109/CLUSTR.2005.347044. URL : http://hal.inria.fr/inria-00070445.

[CI11] Brice GOGLIN et Loïc PRYLLI. « Transparent Remote File Access through a Shared Library Client ». In : *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'04)*. T. 3. Las Vegas, Nevada : CSREA Press, juin 2004, p. 1131–1137. URL : http://hal.inria.fr/inria-00071527.

[CI12] Teng MA et al. « Kernel Assisted Collective Intra-node MPI Communication Among Multi-core and Many-core CPUs ». In : *Proceedings of the 40th International Conference on Parallel Processing (ICPP-2011)*. Taipei, Taiwan, sept. 2011. DOI : 10.1109/ICPP.2011.29. URL : http://hal.inria.fr/inria-00602877.

[CI13] Stéphanie MOREAUD et Brice GOGLIN. « Impact of NUMA Effects on High-Speed Networking with Multi-Opteron Machines ». In : *Proceedings of the 19th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2007)*. Cambridge, Massachussetts : ACTA Press, nov. 2007, p. 24–29. URL : http://hal.inria.fr/inria-00175747.

[CI14] Stéphanie MOREAUD, Brice GOGLIN et Raymond NAMYST. « Adaptive MPI Multirail Tuning for Non-Uniform Input/Output Access ». In : *Recent Advances in the Message Passing Interface. The 17th European MPI User's Group Meeting (EuroMPI 2010)*. Sous la dir. d'Edgar Gabriel RAINER KELLER et Jack DONGARRA. T. 6305. Lecture Notes in Computer Science. Best paper award. Stuttgart, Germany : Springer-Verlag, sept. 2010, p. 239–248. DOI : 10.1007/978-3-642-15646-5_25. URL : http://hal.inria.fr/inria-00486178.

[CI15] Bertrand PUTIGNY, Brice GOGLIN et Denis BARTHOU. « A Benchmark-based Performance Model for Memory-bound HPC Applications ». In : *Proceedings of 2014 International Conference on High Performance Computing & Simulation (HPCS 2014)*. Bologna, Italy, juil. 2014. URL : http://hal.inria.fr/hal-00985598.

Workshops internationaux avec comité de lecture

[WI1] François BROQUEDIS et al. « Dynamic Task and Data Placement over NUMA Architectures : an OpenMP Runtime Perspective ». In : *Evolving OpenMP in an Age of Extreme Parallelism, 5th International Workshop on OpenMP, IWOMP 2009*. Sous la dir. de Barbara Chapman MATTHIAS S. MULLER Bronis R. de Supinski. T. 5568. Lecture Notes in Computer Science. Dresden, Germany : Springer, juin 2009, p. 79–92. ISBN : 978-3-642-02284-5. DOI : 10.1007/978-3-642-02303-3_7. URL : http://hal.inria.fr/inria-00367570.

[WI2] Brice GOGLIN. « Decoupling Memory Pinning from the Application with Overlapped on-Demand Pinning and MMU Notifiers ». In : *CAC 2009 : The 9th Workshop on Communication Architecture for Clusters, held in conjunction with IPDPS 2009*. Rome, Italy : IEEE Computer Society Press, mai 2009. DOI : 10.1109/IPDPS.2009.5160888. URL : http://hal.inria.fr/inria-00356236.

[WI3] Brice GOGLIN. « Design and Implementation of Open-MX : High-Performance Message Passing over generic Ethernet hardware ». In : *CAC 2008 : Workshop on Communication Architecture for Clusters, held in conjunction with IPDPS 2008*. Miami, FL : IEEE Computer Society Press, avr. 2008. DOI : 10.1109/IPDPS.2008.4536140. URL : http://hal.inria.fr/inria-00210704.

[WI4] Brice GOGLIN et Nathalie FURMENTO. « Enabling High-Performance Memory-Migration in Linux for Multithreaded Applications ». In : *MTAAP'09 : Workshop on Multithreaded Architectures and Applications, held in conjunction with IPDPS 2009*. Rome, Italy : IEEE Computer Society Press, mai 2009. DOI : 10.1109/IPDPS.2009.5161101. URL : http://hal.inria.fr/inria-00358172.

[WI5] Brice GOGLIN et Stéphanie MOREAUD. « Dodging Non-Uniform I/O Access in Hierarchical Collective Operations for Multicore Clusters ». In : *CASS 2011 : The 1st Workshop on Communication Architecture for Scalable Systems, held in conjunction with IPDPS 2011*. Anchorage, AK : IEEE Computer Society Press, mai 2011. DOI : 10.1109/IPDPS.2011.222. URL : http://hal.inria.fr/inria-00566246.

[WI6] Brice GOGLIN et Loïc PRYLLI. « Performance Analysis of Remote File System Access over a High-Speed Local Network ». In : *Proceedings of the Workshop on Communication Architecture for Clusters (CAC'04), held in conjunction with the 18th IEEE IPDPS Conference*. Santa Fe, New Mexico : IEEE Computer Society Press, avr. 2004, p. 185. DOI : 10.1109/IPDPS.2004.1303196. URL : http://hal.inria.fr/inria-00071791.

[WI7] Brice GOGLIN, Loïc PRYLLI et Olivier GLÜCK. « Optimizations of Client's side communications in a Distributed File System within a Myrinet Cluster ». In : *Proceedings of the IEEE Workshop on High-Speed Local Networks (HSLN), held in conjunction with the 29th IEEE LCN Conference*. Tampa, Florida : IEEE Computer Society Press, nov. 2004, p. 726–733. DOI : 10.1109/LCN.2004.92. URL : http://hal.inria.fr/inria-00071414.

[WI8] Stéphanie MOREAUD et al. « Optimizing MPI Communication within large Multicore nodes with Kernel assistance ». In : *CAC 2010 : The 10th Workshop on Communication Architecture for Clusters, held in conjunction with IPDPS 2010*. Atlanta, GA : IEEE Computer Society Press, avr. 2010. DOI : 10.1109/IPDPSW.2010.5470849. URL : http://hal.inria.fr/inria-00451471.

[WI9] Bertrand PUTIGNY, Brice GOGLIN et Denis BARTHOU. « Performance modeling for power consumption reduction on SCC ». In : *Proceedings of the 4th Many-core Applications Research Community (MARC) Symposium*. Sous la dir. de Peter TRÖGER et Andreas POLZE. Technical Reports of University of Postdam Hasso Plattner Institute 55. Potsdam, Germany, fév. 2012, p. 21–26. ISBN : 978-3-86956-169-1. URL : http://hal.inria.fr/hal-00649635.

[WI10] Bertrand PUTIGNY, Benoit RUELLE et Brice GOGLIN. « Analysis of MPI Shared-Memory Communication Performance from a Cache Coherence Perspective ». In : *Proceedings of the 15th IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing (PDSEC 2014), held in conjunction with IPDPS*. Phoenix, AZ, mai 2014. URL : http://hal.inria.fr/hal-00956307.

[WI11] Samuel THIBAULT et al. « An Efficient OpenMP Runtime System for Hierarchical Architectures ». In : *A Practical Programming Model for the Multi-Core Era, 3rd International Workshop on OpenMP, IWOMP 2007, Beijing, China, June 3-7, 2007, Proceedings*. Sous la dir. de Barbara M. CHAPMAN et al. T. 4935. Lecture Notes in Computer Science. Springer, 2008, p. 161–172. ISBN : 978-3-540-69302-4. DOI : 10.1007/978-3-540-69303-1_19. URL : http://hal.inria.fr/inria-00154502.

Revues françaises avec comité de lecture

[RF1] Brice GOGLIN, Olivier GLÜCK et Pascale Vicat-Blanc PRIMET. « Interaction efficace entre les réseaux rapides et le stockage distribué dans les grappes de calcul ». In : *Technique et Science Informatiques* 27.7/2008 (sept. 2008), p. 911–940. DOI : 10.3166/tsi.27.911-940. URL : http://hal.inria.fr/inria-00491732.

Conférences françaises avec comité de lecture

[CF1] Brice GOGLIN et al. « Accès optimisés aux fichiers distants dans les grappes disposant d'un réseau rapide ». In : *Actes de RenPar'16, CFSE'4, SympAAA'2005*. Le Croisic, Presqu'île de Guérande, France, avr. 2005, p. 37–46. URL : http://hal.inria.fr/inria-00070548.

Livres

[Li1] Pascale VICAT-BLANC et al. *Computing Networks : From Cluster to Cloud Computing*. ISTE Ltd et John Wiley & Sons Inc., mai 2011, p. 261. ISBN : 978-1-84821-286-2. URL : http://eu.wiley.com/WileyCDA/WileyTitle/productCd-1848212860.html.

[Li2] Pascale VICAT-BLANC et al. *Réseaux de calcul - des grappes aux nuages de calcul*. Collection architecture, applications, service. Hermès Science - Lavoisier, sept. 2010, p. 213. ISBN : 978-2-7462-3006-4. URL : http://www.lavoisier.fr/notice/fr2746230060.html.

Articles de médiation scientifique

[MS1] Brice GOGLIN. « De votre boulangerie à un système d'exploitation multiprocesseur ». Français. In : *Interstices* (fév. 2011). URL : http://hal.inria.fr/inria-00566232.

[MS2] Brice GOGLIN. « Et plus vite si affinités... » Français. In : *Interstices* (juin 2011). URL : http://hal.inria.fr/inria-00604025.

[MS3] Brice GOGLIN. « Les réseaux pour le calcul haute performance : facteur, livreur ou déménageur ? » Français. In : *Interstices* (déc. 2013). URL : http://hal.inria.fr/hal-00915723.

[MS4] Brice GOGLIN et Bertrand PUTIGNY. « Idée reçue : Comparer la puissance de deux ordinateurs, c'est facile ! » Français. In : *Interstices* (avr. 2013). URL : http://hal.inria.fr/hal-00816422.

Autres articles de revues

[RA1] Brice GOGLIN. « What HPC Networking Requires from the Linux Kernel ». In : *HPC Wire, LinuxWorld Conference & Expo Features* (16 août 2006). URL : http://hal.inria.fr/hal-00975608.

[RA2] Brice GOGLIN, Jeff SQUYRES et Samuel THIBAULT. « Hardware Locality : Peering under the hood of your server ». In : *Linux Pro Magazine* 128 (juil. 2011), p. 28–33. URL : http://hal.inria.fr/inria-00597961.

Rapports de recherche

[RR1] Scott ATCHLEY et Brice GOGLIN. *Comparing Lustre Performance using Myricom 10G Dual Protocol NICs*. Research Report. Myricom, Inc., 2006. URL : http://hal.inria.fr/hal-00691977.

[RR2] Brice GOGLIN. *High-Speed Networking in Clusters without OS-bypass and Zero-copy*. Research Report. Myricom, Inc., 2006. URL : http://hal.inria.fr/hal-00691967.

[RR3] Brice GOGLIN. *On the Efficient Usage of High-Speed Networks for Distributed Storage in Clusters*. Research Report. INRIA, 2007. URL : http://hal.inria.fr/inria-00070218.

[RR4] Brice GOGLIN et Loïc PRYLLI. *Design and Implementation of ORFA*. Technical Report TR2003-01. Lyon, France : LIP, ENS Lyon, sept. 2003. URL : http://hal.inria.fr/inria-00408749.

Thèse

[Th1] Brice GOGLIN. « Réseaux rapides et stockage distribué dans les grappes de calculateurs : propositions pour une interaction efficace ». Thèse de doct. 46, allée d'Italie – 69364 Lyon cedex 07 – France : École normale supérieure de Lyon, oct. 2005. URL : http://tel.archives-ouvertes.fr/tel-00408722.

Annexe B
Publications

Voici mes cinq publications les plus significatives. Elles reprennent chacunes un de mes grands axes de recherche.

Improving Message Passing over Ethernet with I/OAT Copy Offload in Open-MX a été publié à la conférence *International Conference on Cluster Computing (IEEE)* en 2008 [CI5]. Contrairement à [RI2] qui présente une vue d'ensemble de mes travaux sur les communications sur réseaux Ethernet (chapitre 2), celle-ci est la publication dont je suis le plus fier dans ce domaine car elle présente une optimisation inédite des communications MPI.

KNEM : a Generic and Scalable Kernel-Assisted Intra-node MPI Communication Framework a été publié dans la revue internationale *Journal of Parallel and Distributed Computing (JPDC, Elsevier)* en 2013 [RI4]. Elle présente tout mon travail sur les communications intra-nœuds en se focalisant sur la conception du logiciel KNEM (chapitre 3). De plus amples détails sur son utilisation dans MPI se trouvent dans [CI12].

hwloc : a Generic Framework for Managing Hardware Affinities in HPC Applications a été publié à la conférence *International Conference on Parallel, Distributed and Network-Based Processing (PDP, EuroMicro)* en 2010 [CI1]. Cet article présente la conception de hwloc et quelques premiers exemples d'utilisation dans différents domaines du HPC étudiés dans l'équipe (chapitre 4). Il a déjà été cité plus de 100 fois.

ForestGOMP : an efficient OpenMP environment for NUMA architectures a été publié dans la revue *International Journal on Parallel Programming (IJPP, Springer)* en 2010 [RI1]. Il présente tous nos travaux sur l'ordonnancement conjoint des threads OpenMP et de la mémoire dans le logiciel ForestGOMP ainsi que les fonctionnalités de migration mémoire avancée dans Linux. (section 5.2).

Adaptive MPI Multirail Tuning for Non-Uniform Input/Output Access a été publié à la conférence internationale *European MPI User's Group Meeting (EuroMPI, Springer)* en 2010 [CI14] où il y a reçu un *Best Paper Award*. Il présente une des deux optimisations des communications réseau en lien avec les architectures à accès non uniforme aux périphériques d'entrées-sorties (NUIOA, section 5.3), l'autre étant présentée dans [WI5].