HAL
open science

Mouvement de données et placement des tâches pour les communications haute performance sur machines hiérarchiques

Stéphanie Moreaud

To cite this version:

Stéphanie Moreaud. Mouvement de données et placement des tâches pour les communications haute performance sur machines hiérarchiques. Réseaux et télécommunications [cs.NI]. Université Sciences et Technologies - Bordeaux I, 2011. Français. NNT : . tel-00635651v3

HAL Id: tel-00635651

https://theses.hal.science/tel-00635651v3

Submitted on 28 Nov 2011

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

N° d'ordre : 4325

THÈSE

présentée à

L'Université Bordeaux 1

École Doctorale de Mathématiques et Informatique

par Stéphanie MOREAUD

pour obtenir le grade de

Docteur

Spécialité : Informatique

Mouvement de données et placement des tâches pour les communications haute performance sur machines hiérarchiques

Soutenue le : 12 Octobre 2011

Après avis de :

M. Franck	CAPPELLO	Directeur de Recherche INRIA, Université de l'Illinois, Urbana Champaign	Rapporteur
M. Bernard	TOURANCHEAU	Professeur, Université Joseph Fourier, Grenoble	Rapporteur

Devant la commission d'examen formée de :

M. Franck	CAPPELLO	Directeur de Recherche INRIA, Université de l'Illinois, Urbana Champaign	Rapporteur
M. Olivier	COULAUD	Directeur de Recherche INRIA	Président
M. Olivier	GLÜCK	Maître de conférences, Université Claude Bernard, Lyon	Examinateur
M. Brice	GOGLIN	Chargé de Recherche INRIA	Directeur de thèse
M. Raymond	NAMYST	Professeur, Université Bordeaux I	Directeur de thèse
M. Bernard	TOURANCHEAU	Professeur, Université Joseph Fourier, Grenoble	Rapporteur

Remerciements

Je voudrais tout d'abord remercier l'ensemble des membres de mon jury, Olivier Coulaud, Olivier Glück, Brice Goglin, Raymond Namyst et tout particulièrement Franck Cappello et Bernard Tourancheau qui ont accepté de relire ce mémoire.

Je souhaite remercier mes deux encadrants, Brice Goglin et Raymond Namyst pour m'avoir guidée depuis mon DEA et tout au long de cette thèse. Cela a été un réel plaisir de travailler avec eux et je n'aurais pu souhaiter meilleurs mentors. Merci à Raymond pour la qualité de ses enseignements et son discours passionné sur la recherche qui m'ont conduit vers le chemin de la thèse, ainsi que pour ses conseils avisés tant sur le plan scientifique que cinématographique. Merci à Brice pour son incroyable patience et sa disponibilité sans lesquelles je n'aurais pu apprendre autant (notamment à parler `lspci` ou à lire dans les tables de routages HYPERTRANSPORT), et pour avoir été ma boussole lorsque je ne savais plus dans quelle direction aller.

Je remercie ensuite tous les membres de l'équipe Runtime qui constituent un noyau de travail et d'échanges aussi riche que sympathique : PAW dont le flegme est légendaire parmi les étudiants ; Olivier et Alexandre pour leur modules de SDRP et leurs talents de photographes ; Emmanuel et Denis qui trouvent toujours à animer les déjeuners-agéco ; BigRay pour la Grande Vadrouille, Tom Cruise ! et les rayures ;-) ; Sam dont les compétences dans de très nombreux domaines ne cessent de m'impressionner ; Nathalie pour les importations allemandes (et pour n'avoir jamais refusé de relire mon anglais) ; Marie-Christine et sa douceur ; Brice qui fait *"collaborer les composants d'un ordinateur"* et porte à merveille LaTeX et masque blanc ! ; Guigui pour Nanarland ; Civodul toujours souriant face à NixOS ; Sylvie sans qui nous serions perdus ; Cédric et Jéjé mes compagnons de rédaction que j'admire d'avoir trouvé le temps de se marier ; Ludovic pour les *potins du chef* et Yannick toujours prêt à en rigoler ; Sylvain qui comprendra bientôt que *"Alors la rédaction ?"* n'est pas la meilleure façon de saluer un doctorant ; Julien pour les discussions métalo-rôlistiques ; Bertrand qui prend le relais ; et tous les "jeunes Runtimers", Andra, Nicolas, Corentin, Sébastien, Cyril, François, etc.
Un grand merci aussi à tous ceux qui sont partis (et qui nous manquent) : Christophe, Sylvain, Babeth et ses TDs de Java, Paulette et les sessions baptême de PIOMan sur irc, Broq dont je me languis de l'humour, Diak mon co-bureau distant, Mateo pour ses conseils de résident aux US, Louis-Claude et les menus végétariens, Rémi qui voit Runtime en musique, Mam'zelle Chocolat et ses ratons, Andrès, Akihiro, Bambi et les stagiaires "tatata McGyver", ...

Merci également à tous les membres du département Info de l'IUT avec lesquels j'ai pris un très grand plaisir à travailler (et discuter au coin café !), à tous ceux que j'ai croisés au cours de mes enseignements, et aux admins du CREMI pour le comparatif *Hagrid* vs *Lebowski*.

Merci également à Virginie d'avoir accueilli l'équipe dans son jardin pour les barbecues, à Emilie d'avoir porté fièrement son collier d'EVJF, et à Melvin, Raphaël, Julien, Marie et Romain d'avoir autorisé leurs Papas à relire une partie de ce pavé à la maison.

Merci à l'ensemble des habitants du A29/A29bis, et notamment à tous les *ex-Scalala* et affiliés pour leur bonne humeur, Cédric et la joyeuse bande du SED pour leur démonstration de football (et la prétention du discours :-D) ainsi que les déjeuners chez Peppino, Séverine pour ces talents de recrutement aux forums/jeux & co, Jean-Philippe pour avoir aidé à lutter contre la pluie d'hiver dans mon bureau, Laetitia pour les échanges culinaires et Magic François pour sa gestion de l'amphi et la session déstress pré-soutenance.

Je remercie enfin ma famille et mes amis pour m'avoir soutenue tout au long de ma thèse et tous ceux qui ont fait le déplacement, parfois de loin pour venir assister à ma soutenance.

Merci à la famille Chabanel pour les tea-parties monpazieroises (et Bénédicte pour les mini-pizzas) ; à Patrick Allaert qui a confectionné Pi 2010 ; à Jennymary et l'atelier théâtre ; à Flo, Pef, Thomas, Drake, Baka, Rémi, Nico et Aya que c'est toujours un plaisir de retrouver ! ; à Bruno parce que Piou ! et à Gaël, dernier bastion de notre groupe à Bordeaux 1 (courage !) ; à Mandine pour nos pauses rédaction et à France Télécom pour les avoir rendues possibles malgré les 600 km qui nous séparaient ; à Guillaume (F***), Gniarf, Fred (super baby-sitter !), Gladys, Yannou et Estelle, qui se sont fait les garants de mes relations sociales (et avec elles ceux de ma santé psychologique) ; à tous les membres de mon Arkana pour avoir réfreiné le *Fanatisme Chaotique* d'Eurydice et à tous leurs joueurs, Yannick, Bibou, Pompon et Steven pour nos interludes rôlistiques.

Merci à tous ceux que je n'ai pas cité personnellement (ou que j'ai oublié :-$).
Je salue le courage de Marie, Sana, Amandine, Nathalie, et ma Maman (qui a tout lu) pour leur passes d'orthographe sur de (très) nombreux chapitres.

Merci à Luc et Maylis, mes parents, qui m'ont encouragée dans tout ce que j'ai pu entreprendre et à qui je dois beaucoup (entre autre la confection de mon très apprécié pot de thèse !), à ma Grand-Mère et à Tante Jo pour leur encouragements, et à mes 3 petits frères, Adrien (ou Minimoi) qui amène toujours de nouvelles anecdotes et de nouveaux records à battre (20 min pour la CB !), Emmanuel pour son infinie gentillesse (et les tartes aux fraises !), et Frédéric dont la malice et la répartie sont toujours rafraîchissantes (1D10 +2 de bonus dans les compétences !). Merci enfin à Kristian qui m'a soutenue au mieux tout au long de ma rédaction malgré la distance (et toléré les restaurants de salades pour nos retrouvailles) et a su trouver les mots pour me motiver et me faire rire (*mon précieux...*).

Merci.

Stéphanie

Résumé : Les architectures des machines de calcul sont de plus en plus complexes et hiérarchiques, avec des processeurs multicœurs, des bancs mémoire distribués, et de multiples bus d'entrées-sorties. Dans le cadre du calcul haute performance, l'efficacité de l'exécution des applications parallèles dépend du coût de communication entre les tâches participantes qui est impacté par l'organisation des ressources, en particulier par les effets NUMA ou de cache.

Les travaux de cette thèse visent à l'étude et à l'optimisation des communications haute performance sur les architectures hiérarchiques modernes. Ils consistent tout d'abord en l'évaluation de l'impact de la topologie matérielle sur les performances des mouvements de données, internes aux calculateurs ou au travers de réseaux rapides, et pour différentes stratégies de transfert, types de matériel et plateformes. Dans une optique d'amélioration et de portabilité des performances, nous proposons ensuite de prendre en compte les affinités entre les communications et le matériel au sein des bibliothèques de communication. Ces recherches s'articulent autour de l'adaptation du placement des tâches en fonction des schémas de transfert et de la topologie des calculateurs, ou au contraire autour de l'adaptation des stratégies de mouvement de données à une répartition définie des tâches.

Ce travail, intégré aux principales bibliothèques MPI, permet de réduire de façon significative le coût des communications et d'améliorer ainsi les performances applicatives. Les résultats obtenus témoignent de la nécessité de prendre en compte les caractéristiques matérielles des machines modernes pour en exploiter la quintessence.

Mots-clés : Calcul intensif, communication réseau, mémoire partagée, MPI, multiprocesseur, NUMA, multicœurs, affinité matérielle, topologie.

Abstract : The emergence of multicore processors led to an increasing complexity inside the modern servers, with many cores, distributed memory banks and multiple Input/Output buses. The execution time of parallel applications depends on the efficiency of the communications between computing tasks. On recent architectures, the communication cost is largely impacted by hardware characteristics such as NUMA or cache effects.

In this thesis, we propose to study and optimize high performance communication on hierarchical architectures. We first evaluate the impact of the hardware affinities on data movement, inside servers or across high-speed networks, and for multiple transfer strategies, technologies and platforms. We then propose to consider affinities between hardware and communicating tasks inside the communication libraries to improve performance and ensure their portability. To do so, we suggest to adapt the tasks binding according to the transfer method and the topology, or to adjust the data transfer strategies to a defined task distribution.

Our approaches have been integrated in some main MPI implementations. They significantly reduce the communication costs and improve the overall application performance. These results highlight the importance of considering hardware topology for nowadays servers.

Keywords : High Performance Computing, network communication, shared memory, MPI, multiprocessor, NUMA, multicore, hardware affinity, topology.

Table des matières

Introduction **ix**

I État de l'art **5**

1 Évolution des architectures parallèles **7**
1.1 L'hégémonie des grappes de calcul 8
1.1.1 Les réseaux haute performance 9
1.1.2 Mécanismes clés des communications sur réseaux rapides 9
1.1.3 Description des principaux réseaux haute performance 13
1.1.4 Bilan 15
1.2 Évolution des nœuds de calcul au sein des grappes 16
1.2.1 La révolution du multicœur 17
1.2.2 Le retour du NUMA 19
1.2.3 Tendances : une complexification grandissante 22
1.3 Bilan : un parallélisme hiérarchique 22

2 Exploiter les architectures parallèles **25**
2.1 Modèles de programmation pour les architectures parallèles 26
2.1.1 Programmation par mémoire partagée 26
2.1.1.1 Multithreading 26
2.1.1.2 Langages de programmation 27
2.1.2 Programmation pour les systèmes à mémoire distribuée 28
2.1.2.1 Passage de messages 28
2.1.2.2 Accès directs à la mémoire distante 29
2.1.3 Mémoire virtuellement partagée 30
2.1.4 Modèles de programmation hybrides 31
2.1.5 Bilan 32
2.2 Gestion des communications dans les bibliothèques MPI 32
2.2.1 Stratégies de transferts utilisées par les bibliothèques MPI 33
2.2.2 Opérations collectives et réduction du trafic interprocessus 36
2.2.3 Affiner les algorithmes de communication 37
2.2.4 Bilan 38
2.3 Impact des architectures contemporaines 39
2.3.1 Quels défis pour les programmeurs ? 39

2.3.2 Quels supports pour détecter et exploiter la topologie des architectures ? . 42
2.3.2.1 Détecter la topologie et forcer le placement des tâches 43
2.3.2.2 Vers une meilleure collaboration des tâches 44
2.3.3 Des conséquences de la topologie sur les communications réseau 46
2.4 Bilan 48

II Contribution 51

3 Vers des communications qui s'adaptent à la topologie 53
3.1 Communication et placement 53
3.2 Affiner les communications sur les grappes à architecture moderne 55
3.3 Ajuster le placement des processus sur les structures matérielles 56
3.4 Discussion 57

4 Des effets NUIOA à l'adaptation du placement pour les communications réseau 61
4.1 Évaluation de l'impact des effets NUIOA sur les communications 62
4.1.1 Plateforme de test 62
4.1.2 Effets NUIOA sur les transferts locaux 64
4.1.3 Répercussion des effets NUIOA sur les communications applicatives . . 67
4.1.4 Spectre d'impact NUIOA 76
4.1.5 Bilan 80
4.2 Implémentation d'une solution de placement automatique 81
4.2.1 Trouver le nœud le plus proche de chaque carte réseau 82
4.2.2 Mise en œuvre dans NEWMADELEINE 82
4.2.3 Cas des communications multirails 83
4.3 Bilan 84

5 Adaptation des communications MPI aux contraintes NUIOA 85
5.1 Adaptation des transferts réseau multirails aux effets NUIOA 86
5.1.1 Stratégie de transfert multirail dans OPEN MPI 86
5.1.2 Implémentation 88
5.1.3 Variation du ratio de division et performances des transferts multirails . . 90
5.1.4 Bilan 95
5.2 Intégration des contraintes NUIOA dans les opérations collectives MPI 95
5.2.1 Collectives hiérarchiques et placement NUIOA 95
5.2.2 Mise en œuvre et performances 100
5.2.3 Bilan 104
5.3 Vers des stratégies de transfert réseau adaptées à la localité des cartes 104

6 Adaptation des communications en mémoire partagée 107
6.1 Communications en mémoire partagée dans MPICH2 108
6.1.1 Transferts de larges messages dans MPICH2-NEMESIS 108
6.2 Analyse des performances 114
6.2.1 Plateforme de test et méthodologie 114
6.2.2 Communications point-à-point 116

6.2.3 Opérations collectives 124
6.2.3.1 Performance des LMTs 124
6.2.3.2 Étude des seuils de transition 126
6.2.4 NAS Parallel Benchmarks 129
6.3 Bilan 130

Conclusion **133**

A Plateforme de tests **141**
A.1 Dalton/Dalton-Infini 141
A.2 Hagrid 142
A.3 Dancharia 143
A.4 Borderline 144
A.5 Mirage 144
A.6 Hannibal 145
A.7 Bill 146
A.8 Idkonn 146
A.9 Bertha 147

B Performances des transferts de MPICH2-NEMESIS en mémoire partagée **149**
B.1 Performances des modes synchrone et asynchrone de KNEM 149
B.2 Comparaison des LMTs pour les transferts point-à-point 150
B.3 Performances des opérations collectives 151
B.3.1 Opérations *all-to-one* 151
B.3.2 Opérations *one-to-all* 152
B.3.3 Opérations *all-to-all* 152
B.4 Performances applicatives des différents LMTs pour les tests NAS 152

Table des figures

1.1 Évolution de l'architecture des calculateurs répertoriés au Top500. 9
1.2 Transfert d'un bloc de données en mode PIO. 11
1.3 Transfert des données en mode DMA. 11
1.4 Allure des courbes de coût de transfert en fonction du mode utilisé. 11
1.5 Protocole de Rendez-Vous. 12
1.6 Évolution des technologies réseau dans les machines du Top500. 15
1.7 Architecture SMP. 16
1.8 Processeur sans cœur. 17
1.9 Puce bi-cœur. 17
1.10 Puce hexa-cœur INTEL Xeon Dunnington X7460. 18
1.11 Architecture SMP multicœurs. 19
1.12 Architecture NUMA multicœur à quatre nœuds. 19
1.13 Processeur Opteron. 21
1.14 Quadri-processeur AMD Opteron quadri-cœur Barcelona 21

2.1 Débits de ping-pong au-dessus de OPEN MPI. 33
2.5 Techniques de communication intra-nœud. 34
2.6 Schémas de communications collectives. 36
2.7 Communication one-to-all sur deux nœuds disposant de 4 processus chacun. . . 38
2.8 Topologie NUMA de la machine Dancharia. 40
2.9 Débit de lecture mémoire en fonction de la position des données 41
2.10 Processeur Intel Xeon hexa-cœur Dunnington. 42
2.11 Exemple de sortie graphique de `lstopo`. 45
2.12 Débits de ping-pong multirails sur NEWMADELEINE en fonction du placement. . 48

3.1 Représentation du trafic interprocessus en fonction du placement des tâches. . . . 54
3.2 Champs d'intervention pour prendre en compte de la hiérarchie des nœuds de calcul. 59

4.1 Topologies NUMA et réseau de notre plateforme de test OPTERON. 62
4.2 Topologies NUMA et réseau de notre plateforme de test Intel. 63
4.3 Trafic sur les liens d'interconnexion pour un transfert de données entre deux hôtes. 64
4.4 Impact du placement NUMA sur la latence des transferts locaux. 65
4.5 Débits de ping-pong multirails en fonction du placement. 68
4.6 Débits de ping-pong monorails en fonction du placement NUMA. 69
4.7 Débits de ping-pong OPEN MPI en fonction du placement sur les Dalton. 70

4.8 Débits de ping-pong OPEN MPI en fonction du placement sur Borderline. 71
4.9 Débits de ping-pong OPEN MPI en fonction du placement sur Mirage. 71
4.10 Variation du débit des transferts RDMA en fonction du placement NUMA. 72
4.11 Performances de transferts unidirectionnels en fonction du placement NUMA. . . 72
4.12 Débits de transferts RDMA en fonction du placement sur Mirage. 73
4.13 Débits de transferts RDMA en fonction du placement sur Hannibal. 74
4.14 Débits de transferts RDMA en fonction du placement sur Hagrid. 75
4.15 Perturbation d'un RDMA par des écritures mémoire. 75
4.16 Architectures NUMA et NUIOA. 77
4.17 Structure des processeurs Sandy-Bridge attendue dans les serveurs INTEL. 77
4.18 Impact NUIOA sur les transferts entre mémoire et GPU sur Hannibal. 78
4.19 Bi quadri-cœur Nehalem doté de 3 GPUs NVIDIA. 78
4.20 Impact NUIOA sur les transferts entre mémoire et GPU sur Mirage. 79
4.21 Bi hexa-cœur Westmere doté de 3 GPUs NVIDIA. 79
4.22 Architecture avec deux contrôleurs d'entrées-sorties pourvus d'I/OAT. 80
4.23 Débits des transferts de données en fonction de la localité du moteur DMA. . . . 80

5.1 Exemple de distribution des tâches en fonction des besoins applicatifs. 87
5.2 Exemple de transferts multirails. 88
5.3 Représentation partielle de la pile logicielle OPEN MPI. 88
5.4 Transfert multirail au travers de la pile OPEN MPI. 89
5.5 Topologie NUMA des machines Borderline. 90
5.6 Débits de ping-pong monorails OPEN MPI sur Borderline. 90
5.7 Débits de ping-pong IMB multirails fonction du ratio de division et du placement. 91
5.8 Ratio de division multirails dérivé des débits monorails. 92
5.9 Effet de la contention sur le débits de ping-pong multirails. 93
5.10 Débit du test IMB all-to-all en fonction du ratio de division. 94
5.11 Trafic sur les liens mémoire en fonction du placement NUMA. 96
5.12 IMB Bcast entre 8 nœuds sur Borderline en fonction du placement (8 processus). 98
5.13 IMB Bcast entre 8 nœuds sur Mirage en fonction du placement (8 processus). . . 98
5.14 IMB Gather entre 8 nœuds sur Borderline en fonction du placement (8 processus). 99
5.15 IMB Gather entre 8 nœuds sur Mirage en fonction du placement (8 processus). . 99
5.16 Schéma de communications lors d'un broadcast hiérarchique. 100
5.17 Schéma de communications lors d'un broadcast hiérarchique NUIOA. 100
5.18 IMB Bcast pour différents algorithmes hiérarchiques sur Borderline (64 processus). 102
5.19 IMB Bcast pour différents algorithmes hiérarchiques sur Mirage (96 processus). . 102
5.20 IMB Bcast pour le module Tuned sur Borderline (64 processus). 102
5.21 Performances des implémentations de broadcast en fonction du nombre de processus par nœud. 103
5.22 Performances des implémentations de broadcast en fonction du nombre de nœud. 103

6.1 Représentation partielle de la pile logicielle de MPICH2. 109
6.2 Transferts de données à l'aide d'un tampon en en mémoire partagée. 109
6.3 Transfert de données classique au travers d'un tube UNIX. 110
6.4 Transfert au travers d'un tube appuyé sur l'appel système `vmsplice`. 110
6.5 Transfert de larges messages avec le module noyau KNEM. 112

6.6 Détection de la terminaison d'une copie I/OAT par double scrutation dans KNEM 113
6.7 Détection de la terminaison d'une copie I/OAT par scrutation simple dans KNEM 113
6.8 Représentation des processeurs de la machine Bill par `hwloc`. 115
6.9 Représentation des processeurs de la machine Hannibal par `hwloc`. 115
6.10 Représentation des sockets des machines Idkonn et Bertha par `hwloc`. 115
6.11 Effet de l'option offcache sur débit des transferts avec cache partagé. 116
6.12 Effet de l'option offcache sur débit des transferts sans cache partagé. 116
6.13 Pingpong IMB entre 2 processus partageant un cache L2 de 4 Mo. 117
6.14 Pingpong IMB entre 2 processus ne partageant aucun cache. 117
6.15 Débits de ping-pong avec différents LMTs en fonction du placement sur Bertha. . 120
6.16 Pingpong IMB entre 2 processus partageant un cache L2 de 4 Mo. 121
6.17 Pingpong IMB entre 2 processus ne partageant aucun cache. 121
6.18 Pingpong entre 2 processus partageant un cache sur Idkonn. 122
6.19 Seuil de transitions entre Nemesis et le LMT KNEM. 123
6.20 Débits d'un Alltoall IMB sur la machine Bill. 125
6.21 Débits du test Reduce sur Bertha (64 processus). 125
6.22 Débits du test Reduce sur Hannibal (16 processus). 125
6.23 Débits du IMB Scatter sur Idkonn (16 processus). 127
6.24 Débits du IMB Alltoall sur Idkonn (16 processus). 127
6.25 Débits du IMB Alltoall sur Bertha (64 processus). 127

A.1 Bi-processeur bi-cœurs OPTERON Dalton. 141
A.2 Bi-processeur bi-cœurs OPTERON Dalton-Infini. 141
A.3 Octo bi-cœur OPTERON Hagrid. 142
A.4 Topologie NUMA de la machine Dancharia. 143
A.5 Représentation d'un nœud NUMA de la machine Dancharia par `hwloc`. 143
A.6 Quadri bi-cœur OPTERON Borderline. 144
A.7 Bi hexa-cœur INTEL Mirage. 144
A.8 Quadri-cœur INTEL Nehalem Hannibal. 145
A.9 Topologie de la machine Hannibal exposée par `hwloc`. 145
A.10 Topologie de Bill exposée par la bibliothèque `hwloc`. 146
A.11 Topologie d'Idkonn exposée par la bibliothèque `hwloc`. 146
A.12 Connexion de 4 disques sur la machine 4 nœuds Bertha 147
A.13 Topologie de Bertha exposée par la bibliothèque `hwloc`. 148

B.1 Performances du LMT KNEM en mode synchrone et asynchrone sur Bill. 149
B.2 Pingpong IMB (offcache) entre 2 processus partageant un cache sur Bill. 150
B.3 Pingpong IMB (offcache) entre 2 processus sans cache commun sur Bill. 150
B.4 Pingpong IMB (offcache) entre 2 processus partageant un cache sur Hannibal. . . 151
B.5 Pingpong IMB entre 2 processus sur des nœuds NUMA distincts sur Hannibal. . 151
B.6 Débits d'un IMB Reduce sur Bill (8 processus). 151
B.7 Débits d'un IMB Allreduce sur la Idkonn (16 processus). 151
B.8 Débits d'un IMB Scatter sur Bill (8 processus). 152
B.9 Débits d'un IMB Bcast sur Hannibal (8 processus). 152
B.10 Débits d'un IMB Allgather sur Bill (8 processus). 153
B.11 Débits d'un IMB Allgather sur Idkonn (16 processus). 153

Liste des tableaux

2.1 Bande passante de copies mémoire en fonction de la position des données. 40

4.1 Impact du placement NUMA sur la latence des transferts locaux. 64
4.2 Impact du placement NUMA sur le débit des transferts locaux dans les machines Opteron. 66
4.3 Impact du placement NUMA sur le débit des transferts locaux dans les machines Intel 67
4.4 Impact des écritures mémoire sur le débit des transferts RDMA. 75

5.1 Résumé de l'impact du placement NUIOA des leaders. 97
5.2 Résumé de l'impact du placement NUIOA sur les transferts réseau pour des opérations collectives. 99

6.1 Caractéristiques des machines de notre plateforme de test. 115
6.2 Résumé de la comparaison des LMTs double-buffering et KNEM. 119
6.3 Temps d'exécution des NAS Parallel Benchmarks. 129
6.4 Défauts de cache pour le test IS en fonction des LMTs. 129

B.1 Temps d'exécution de différents tests NAS sur Bill. 153
B.2 Défauts de cache en fonction des LMTs. 154

Introduction

"Recent progress towards a complete understanding of the universe has been impressive, but many puzzles remain, cosmology is now a precise science, and we need supercomputers to calculate what our theories of the early universe predict and test them against observations of the present universe."

Stephen Hawking

Du calcul scientifique aux architectures parallèles contemporaines

Les résultats spectaculaires de la simulation numérique ont métamorphosé la recherche dans de nombreux domaines scientifiques tels que la météorologie, la cosmologie, l'aérodynamique, la sismologie, la mécanique des fluides et tant d'autres. Comme en témoigne le *Falcon 7X* de la société *Dassault Aviation* [1] entièrement conçu par filière numérique, il est aujourd'hui possible de développer un avion sans même avoir recours à un prototype ou une maquette. La simulation de *crash tests* permet aux constructeurs automobiles d'effectuer à moindre coût des batteries de tests. BMW réalise ainsi un millier de crashs virtuels par an et par modèle, disposant ainsi de résultats probants sans augmenter les délais de développement, mais aussi de niveaux d'analyse inaccessibles jusque-là. La simulation numérique est également une clé majeure pour retracer la création de l'univers. Grâce à un nouveau modèle et après quelques mois de calcul, les chercheurs d'une équipe de l'observatoire de la Côte d'Azur ont ainsi réussi à expliquer l'origine du bombardement météoritique tardif de la Lune [2], la présence de certains astéroïdes sur l'orbite de Jupiter [3] et la position actuelle des planètes géantes [4].

Au même titre que l'expérimentation, la simulation est ainsi devenue un support indispensable à la recherche scientifique. Elle repose sur la mise en œuvre de modèles théoriques pour modéliser des phénomènes physiques et leurs interactions dans le but de prédire des évènements résultants. Alors que les modèles se précisent, la puissance de calcul nécessaire à leur résolution ne cesse d'augmenter. Les calculs scientifiques requièrent en effet de traiter des problèmes de plus en plus complexes afin d'obtenir des résultats toujours plus précis, sans renoncer à des contraintes de temps d'exécution raisonnables. Pour répondre à cette demande grandissante de performance, les centres de calcul se sont tournés vers l'utilisation du *calcul parallèle* et se sont équipés de machines toujours plus puissantes, suscitant une véritable course à la performance entre les constructeurs.

En 30 ans, le domaine du calcul haute performance a subi plusieurs révolutions matérielles. Les ordinateurs ont évolué vers des architectures multiprocesseurs et des super-calculateurs dédiés qui

ont atteint leur âge d'or dans les années 80. Grâce au développement de nouvelles technologies réseau dans les années 90, les super-calculateurs très onéreux ont cédé leur monopole aux grappes de calcul, composées de plusieurs machines "standard" (nœuds) collaborant et communiquant par le biais de réseaux rapides, et dont le rapport performance/prix s'avère plus abordable. La popularité des grappes a cependant introduit un bouleversement des techniques de programmation et la mise en place de schémas de programmation spécifiques aux architectures distribuées.

Alors que l'augmentation de la puissance des grappes s'appuie sur le développement de la puissance intrinsèque aux nœuds de calcul, les simples PCs initialement employés dans ces architectures distribuées ont évolué vers des machines dotées de multiples processeurs, de plus en plus nombreux et performants, agencés selon diverses topologies internes. Pendant longtemps, la puissance des processeurs a été principalement portée par l'augmentation de leur fréquence. Aujourd'hui, les constructeurs se heurtent à des problèmes de dissipation thermique qui bornent ces fréquences. La solution proposée a conduit à l'avènement du multicœur, la miniaturisation rendant possible la mise en place de plusieurs unités de calcul (cœurs) au sein d'une même puce. Ces puces multicœurs sont désormais le standard des machines dites "de bureau", dont la puissance équivaut aujourd'hui à celles des super-calculateurs d'il y a 15 ans. La programmation parallèle n'est ainsi plus réservée aux seules applications scientifiques, mais s'inscrit désormais dans le cadre des logiciels grand public.

La multiplication des unités de calcul s'accompagne de l'augmentation des ressources qu'elles partagent et amène de nouveaux niveaux de hiérarchie au sein des machines. Les cœurs peuvent ainsi partager des hiérarchies de caches, de multiples interfaces réseau, des bancs mémoire, etc. Les nœuds de calcul modernes sont ainsi constitués de hiérarchies de composants matériels à la manière de véritables poupées russes, et dont l'organisation est à l'origine d'affinités plus ou moins fortes entre les composants. Sur de telles machines, l'utilisation naïve d'applications parallèles dotées de modèles de programmation uniformes et indépendants des structures sous-jacentes entrave les performances tributaires des affinités créées par la topologie matérielle. Exploiter la quintessence de ces architectures est un problème complexe, assujetti à l'adéquation entre les schémas applicatifs, et la structure hiérarchique de l'architecture matérielle. Tandis que les performances théoriques des architectures parallèles s'envolent, le fossé avec les puissances soutenues se creuse, conséquence directe d'une projection inadéquate des structures applicatives sur les structures matérielles. Pour exemple, la grappe Tera-100, entrée en novembre 2010 en sixième position des machines les plus puissantes répertoriées par le Top500 [5], dispose d'une puissance théorique supérieure à un petaflop afin de répondre à la centaine de teraflops *utiles* (dix fois moins) réellement nécessaire au programme de simulation du CEA.

Objectifs et contributions : lier affinités matérielles et communication

Une prise en compte minutieuse de la structure des architectures contemporaines est ainsi un prérequis de performance. Alors que le nombre de tâches participant à l'exécution d'une application parallèle augmente avec la multiplication des unités de calcul disponibles, les communications et synchronisations intervenant entre les tâches ont plus que jamais un rôle critique et leur coût est un facteur déterminant sur les performances obtenues. Les affinités entre les tâches collaborantes et les communications devraient ainsi être prises en compte, sans pour autant gan-

grener les critères de portabilité indispensables à la pérennité des applications parallèles. L'idéal serait ainsi de pouvoir confier la gestion des spécificités matérielles aux bibliothèques de communication ou à l'environnement d'exécution, pour assurer une bonne exploitation de la structure hiérarchique des architectures et la portabilité de performances satisfaisantes.

C'est dans ce contexte que s'inscrivent les travaux présentés dans cette thèse. Ils ont pour but d'introduire une prise en compte de la hiérarchie des machines de calcul modernes dans les problématiques de communication entre les tâches collaborantes à l'exécution d'une application parallèle. Dans le contexte des communications sur grappes de calcul, la topologie des nœuds de calcul peut intervenir au niveau des échanges effectués au travers du réseau (*communication inter-nœuds*) ou des échanges entre des tâches locales à un même nœud (*communication intra-nœud*). Les performances de ces mouvements de données dépendent de la localisation physique des tâches communicantes et des données sur la structure matérielle sous-jacente. Elles sont impactées par les caractéristiques matérielles telles que les effets NUMA, le partage de cache, la localité des cartes réseau, etc. Nos travaux s'articulent ainsi d'abord autour d'une évaluation de l'impact de la hiérarchie des architectures modernes sur les performances des transferts entre les tâches. Puis ils ciblent des optimisations possibles au sein des bibliothèques de communication pour prendre en considération les affinités entres les tâches et les communications, en accord avec la topologie des architectures modernes. La gestion de ces affinités est possible en adaptant le placement des tâches et des données sur la topologies pour favoriser l'adéquation entre les affinités logicielles et matérielles. En complément, ce sont également les techniques de communication qui peuvent être adaptées à un placement prédéfini pour minimiser les pénalités issues de ces caractéristiques. Les deux principaux axes étudiés dans ce document reposent ainsi sur :

- L'adaptation du placement des tâches de communication en fonction des affinités matérielles avec les interfaces réseau.
- L'ajustement des stratégies mises en œuvre dans les bibliothèques de communication aux caractéristiques topologiques des architectures contemporaines.

Organisation du document

Pour appréhender les particularités matérielles caractéristiques des grappes de calcul modernes, le Chapitre 1 retrace l'évolution des architectures parallèles utilisées pour la résolution de problèmes scientifiques. Il présente les progrès accomplis dans le domaine des réseaux haute performance qui ont permis l'émergence des grappes, ainsi que les transformations apportées à la topologie interne des nœuds de calcul. Le Chapitre 2 s'attache à la description des modèles de programmation mis en œuvre pour exploiter ces architectures et inclut une présentation des différents protocoles de transfert et optimisations algorithmiques mis au point au sein des bibliothèques MPI (*Message Passing Interface*). Il décrit également les principaux critères de performance que l'on peut extraire des architectures contemporaines. Le troisième chapitre présente un état de l'art des efforts faits pour optimiser les performances des transferts entre les tâches collaborantes invoquées dans le cadre d'applications MPI sur les clusters de calcul contemporains. Il expose les travaux existants considérant les aspects hiérarchiques des calculateurs et discute des aspects pouvant être traités dans ce domaine.

Les Chapitres 4, 5 et 6 détaillent les contributions et résultats issus de cette thèse. Le Chapitre 4

présente une évaluation approfondie de l'impact de la topologie des machines sur les communications réseau. L'optimisation du placement des tâches en accord avec les contraintes topologiques que nous avons réalisées au sein de la bibliothèque NEWMADELEINE est ensuite détaillé. Le Chapitre 5 considère un placement des tâches contraint dans le cadre d'applications MPI. Il présente les manœuvres possibles permettant d'ajuster l'utilisation de multiples interfaces réseau, puis d'opérations collectives hiérarchiques pour contrebalancer les effets de la topologie sur les communications réseau. Le Chapitre 6 concentre les efforts effectués pour améliorer les communications entre des tâches locales à un même nœud de calcul. Alors que les bibliothèques MPI possèdent des méthodes de transferts diverses, ces dernières varient en termes d'utilisation de ressources et ne sont pas uniformément impactées par les caractéristiques matérielles. Nous avons donc cherché à estimer l'impact topologique sur les différentes stratégies et proposons d'harmoniser leur utilisation conjointe en fonction des spécificités des applications, communications et architectures sous-jacentes. Enfin, ce document conclut sur l'ensemble des travaux menés et discute des pistes de recherches pouvant les étendre.

Première partie

État de l’art

Chapitre 1

Évolution des architectures parallèles

Sommaire

1.1 L'hégémonie des grappes de calcul 8
1.1.1 Les réseaux haute performance 9
1.1.2 Mécanismes clés des communications sur réseaux rapides 9
1.1.3 Description des principaux réseaux haute performance 13
1.1.4 Bilan 15
1.2 Évolution des nœuds de calcul au sein des grappes 16
1.2.1 La révolution du multicœur 17
1.2.2 Le retour du NUMA 19
1.2.3 Tendances : une complexification grandissante 22
1.3 Bilan : un parallélisme hiérarchique 22

Les progrès remarquables issus de la simulation numérique sont le fruit de modèles complexes et précis dont l'étude requiert des puissances de calcul gigantesques. Pour satisfaire ces applications, toujours plus gourmandes en ressources, les constructeurs ont dû améliorer la puissance des machines cibles du calcul intensif. Poussé par la quête de performance des centres de calcul, le calcul haute performance (HPC, *High Performance Computing*) a fait l'objet de recherches et de développements nombreux. En 30 ans, les machines dédiées au calcul intensif ont été marquées par de véritables révolutions technologiques. Ces transformations sont intervenues à de multiples niveaux : amélioration et multiplication des processeurs ; développement de réseaux efficaces ; distribution de la mémoire et production de nouvelles technologies d'interconnexion ; ou encore miniaturisation engendrant l'augmentation du nombre de composants. Les grappes de calcul modernes (*clusters)* employées dans le HPC résultent de ces avancées et ont hérité de caractéristiques matérielles complexes. Pour comprendre les spécificités architecturales intervenant dans la conception des machines de calcul contemporaines, ce chapitre présente les principales technologies qui ont marqué l'évolution du paysage du calcul intensif. Il retrace dans une première partie les avancées liées aux réseaux haute performance et les mécanismes clés de leur exploitation qui ont permis l'émergence des grappes. Il détaille ensuite l'évolution matérielle intrinsèque aux nœuds de calcul, support de la puissance des clusters.

1.1 L'hégémonie des grappes de calcul

Pour assurer le calcul de problèmes scientifiques complexes, les chercheurs n'ont eu d'autre choix que de se tourner vers le parallélisme, décomposant les calculs complexes en plusieurs tâches dont la résolution peut être confiée à de multiples unités de calcul. Ce travail collaboratif exécuté sur architectures parallèles permet ainsi de réduire le temps de calcul nécessaire à la résolution d'un problème. Le raffinement des modèles théoriques, la croissance des données traitées et les besoins de précisions toujours plus poussés ne cessent d'augmenter la puissance de calcul et la mémoire nécessaires au calcul scientifique. La demande croissante de ressources est ainsi à l'origine d'une véritable course à la performance. Au cours des dernières décades, celle-ci a métamorphosé le paysage du calcul intensif.

Supports au calcul parallèle d'ampleur, les *super-calculateurs* ont fait leur entrée dans les centres de calcul des années 70, et connus leur apogée dans les années 80. Ces architectures, dont la conception est propre aux divers constructeurs, sont caractérisées par un parallélisme interne [1] et sont aujourd'hui dotées de plusieurs milliers d'unités de calcul. Elles ont constitué le fleuron de la puissance informatique et représentaient, il y a 12 ans, 95% des machines de calcul les plus puissantes, répertoriées dans le Top500 [5]. Si les super-calculateurs massivement parallèles occupent désormais une place moindre, ces architectures spécifiques n'en restent pas moins performantes. Les 17% des machines qu'elles occupent actuellement au Top500 représentent à elles seules plus de 32% de la puissance de calcul des machines répertoriées. Parmi les 10 premières machines listées se trouvent 5 super-calculateurs [2], notamment la machine *Tianhe-1A* du Centre National de Calcul de Tianjin (Chine), construite par le NUDT (*National University of Defense Technology*), qui occupe actuellement la seconde place du Top500. Inconvénient majeur, les super-calculateurs nécessitent des coûts de développement prohibitifs qui les réservent à un marché très élitiste.

Dans les années 90, grâce à la mise au point de nouvelles technologies réseau, l'architecture des calculateurs a pris un tournant radical, passant d'un modèle massivement parallèle à un modèle distribué. L'engouement pour les super-calculateurs, puissants mais onéreux, s'est estompé au profit des grappes de calcul (*clusters*), comme en témoigne l'évolution des architectures du Top500 illustrée en Figure1.1. Contrairement aux super-calculateurs spécialisés, les grappes se basent sur un parallélisme externe : plutôt que de multiplier les unités de calcul à l'intérieur de la machine, elles font collaborer un ensemble d'ordinateurs indépendants *(nœuds)* qui communiquent au travers d'un réseau dédié.

L'utilisation de grappes de calcul dans le calcul scientifique a émergé il y a une quinzaine d'années avec les projets NOW [25] et Beowulf [24] qui assemblaient des centaines de postes de travail usuels en de larges systèmes. Ce concept a ensuite pris de l'ampleur grâce au développement de nouvelles technologies réseau efficaces. En effet, l'utilisation de réseaux "classiques" implique un coût d'interconnexion entre les processeurs important et constituait un inconvénient considérable par rapport aux architectures massivement parallèles. Le développement des *réseaux haute performance* (Section 1.1.1) a apporté une augmentation du débit notable et une forte réduction de la latence. Les réseaux rapides ont ainsi réduit les coûts de communication sur grappes, permettant l'utilisation des systèmes distribués pour le calcul intensif. Les clusters pouvant alors rivaliser avec

1. Créé par la présence de multiples unités de calcul au sein même de la machine.
2. En juin 2011, les rangs 2, 3, 6, 7 et 8 sont occupés par des super-calculateurs massivement parallèles.

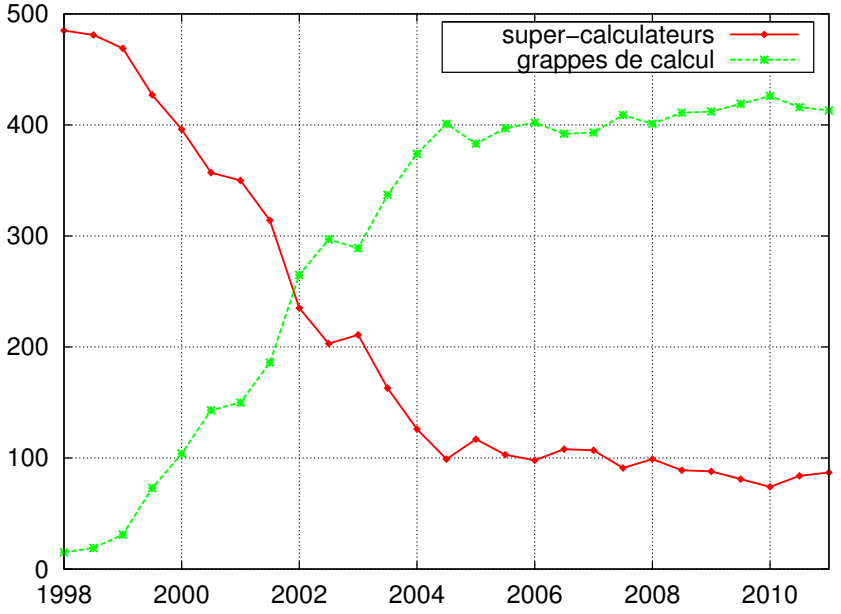


FIGURE 1.1 – *Évolution de l'architecture des calculateurs répertoriés au Top500 [5].*

la puissance déployée par les super-calculateurs, leur rapport performance/prix inégalable porté par l'utilisation de machines relativement standard a assuré leur succès. La place des grappes de calcul dans le champ du calcul intensif n'a ainsi cessé de croître jusqu'à prendre une position largement dominante : en 2011, elles représentent ainsi près de 83% des machines répertoriées dans le Top500 [5] (Figure 1.1).

1.1.1 Les réseaux haute performance

Les applications parallèles nécessitent de nombreux transferts de données et de fréquentes synchronisations entre les processus collaborants dont l'impact sur les performances est crucial. La réduction des temps de communication a ainsi fait l'objet de nombreuses recherches et conduit à l'émergence de nouvelles technologies réseau développées spécifiquement pour le calcul intensif sur grappes de calcul. Les réseaux classiques de type Ethernet ont été remplacés par des réseaux dits *haute performance* ou *rapides*, tels que MYRI-10G [51], QUADRICS [47] ou INFINIBAND [48]. Ceux-ci exhibent de très bonnes performances avec des latences de l'ordre de la microseconde et des débits qui atteignent plusieurs Gigaoctets par seconde.

1.1.2 Mécanismes clés des communications sur réseaux rapides

Au niveau de performance soutenu par les réseaux rapides, le moindre surcoût a son importance. Les bibliothèques de communications employées doivent ainsi effectuer des transferts particulièrement efficaces pour éviter de détériorer les performances exhibées par les technologies

réseau. Pour atteindre cet objectif, elles appliquent des protocoles simplifiés, utilisent différents modes de transfert et réduisent au maximum le chemin critique emprunté par les données.

Protocoles de communication simplifiés et réduction du chemin critique
Un élément important à prendre en compte dans le coût des communications est le protocole de communication utilisé, qui inclut souvent un certain nombre d'étapes garantissant le bon fonctionnement des échanges, l'acheminement et l'intégrité des données. C'est par exemple le cas du protocole TCP/IP qui occupe aujourd'hui une position dominante. Conçu pour Internet, il embarque des mécanismes palliant le manque de fiabilité intrinsèque aux communications longues distances (tolérance aux pannes et aux pertes, contrôle de congestion,...), au prix d'un coût de traversée de la pile logicielle TCP/IP important.

Dans les grappes de calcul, statiques et géographiquement localisées, les niveaux de pérennité et de fiabilité sont élevés et de tels mécanismes peuvent être accessoires. L'utilisation de réseaux rapides s'accompagne ainsi de l'emploi de protocoles de communication simplifiés, soutenu par un ensemble de procédés logiciels permettant de s'affranchir au maximum de tout surcoût. L'idée générale est de réduire au mieux le chemin critique des données entre les processus qui s'exécutent sur des nœuds différents. Les périphériques réseau modernes sont programmables et dotés de processeur et de mémoire embarqués. Il est ainsi possible de confier à l'interface réseau le traitement partiel ou complet du protocole. Le rôle de l'application se restreint alors à la soumission d'une requête à la carte qui prend en charge la gestion du transfert en arrière-plan, permettant le recouvrement d'une partie de la communication par du calcul.

Routage prédéfini
Alors que TCP/IP s'accompagne d'un routage dynamique pour assurer la tolérance aux pannes, la régularité et la staticité des réseaux des grappes permettent d'utiliser un routage prédéfini. Il est ainsi possible d'effectuer un routage à la source et de bénéficier de l'économie du calcul de routage. Sur certaines interfaces, le processeur embarqué sur la carte réseau place les informations de routage en début de paquet. Le travail des commutateurs se réduit alors à extraire cet en-tête et à diriger le paquet.

Modes de transfert entre carte et mémoire
Les communications réseau se font par transfert de données de la mémoire vers l'interface (carte) réseau, par transmission de ces données sur le réseau jusqu'à la carte de la machine distante, et enfin par transfert depuis ce périphérique jusqu'à la mémoire locale. Les échanges de données intervenant entre la mémoire et la carte ont un impact non négligeable sur les performances des communications, notamment sur le débit.

Ces transferts locaux, entre carte réseau et la mémoire, peuvent être effectués selon deux méthodes : en mode PIO (*Programmed Input/Output*) ou en mode DMA (*Direct Memory Access*). L'utilisation des entrées-sorties en mode PIO est faite à l'initiative du processeur. Le transfert des messages est généralement découpé en transferts de blocs de quelques octets. Pour chacun d'entre eux, le processus lit le bloc en mémoire avant de le transférer vers la carte réseau, comme illustré en Figure 1.2. Ce protocole est très efficace pour des données de petite taille, mais souffre de la

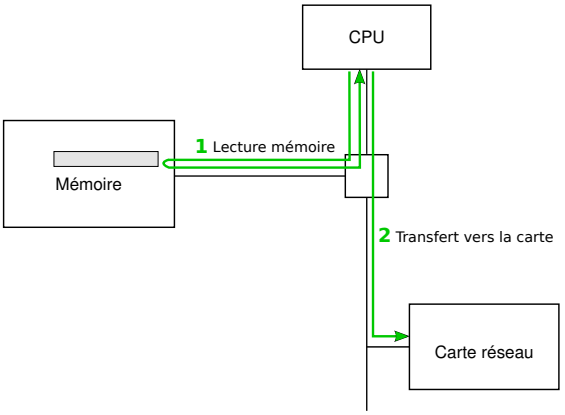


FIGURE 1.2 – *Transfert d'un bloc de données en mode PIO.*

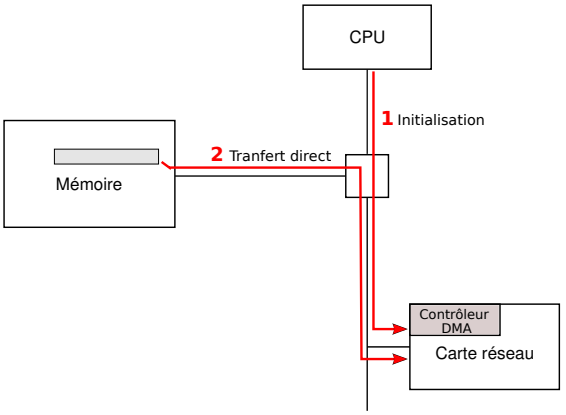


FIGURE 1.3 – *Transfert des données en mode DMA.*

consommation excessive du processeur et se traduit par une monopolisation de celui-ci pour le transfert de grands volumes de données. Le mode DMA permet d'augmenter le débit de la copie des données et son recouvrement par du calcul. Il s'appuie sur l'utilisation d'un contrôleur externe au processeur (Figure 1.3). Le processeur envoie une requête au contrôleur DMA de la carte réseau (1), la copie des données est ensuite effectuée en arrière-plan (2), laissant le processeur (qui n'est plus sollicité jusqu'à la réception d'une interruption lui notifiant la fin du transfert) libre pour effectuer des calculs. Ce mécanisme réduit fortement l'occupation processeur mais nécessite une étape d'initialisation avec un surcoût constant et assez important.

En terme de consommation processeur, il est donc préférable d'utiliser le mode DMA à partir d'une certaine taille de données. En terme de latence, le temps d'initialisation très court du mode PIO le rend plus performant pour les petits messages. Les bibliothèques de communication utilisent donc généralement un combiné de ces deux modes, utilisant des transferts PIO pour les données de petite taille, et DMA pour des données plus larges (Figure 1.4).

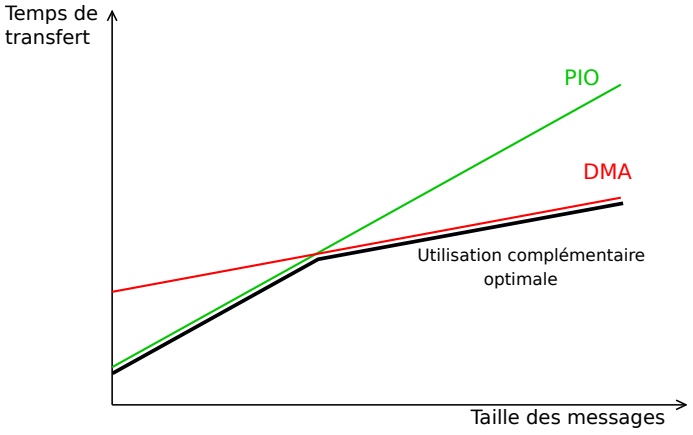


FIGURE 1.4 – *Allure des courbes de coût de transfert en fonction du mode utilisé.*

Stratégies d'envoi et de notification

Lors du transfert des données entre la carte et la mémoire, les protocoles de communication peuvent être à l'origine de recopies de données (ajout d'en-têtes et regroupement de données en émission, utilisation d'un tampon intermédiaire en réception). De telles copies peuvent êtres nécessaires par exemple pour prendre en charge l'arrivée d'un message, qui sera alors stocké dans un tampon intermédiaire le temps que le récepteur détermine un emplacement mémoire destination dans lequel seront recopiées les données. Elles ont cependant un coût proportionnel à la taille des données et augmentent la latence effective des transferts, nécessitant de les restreindre au maximum. Le transfert de données de grande taille peut ainsi être assisté par l'utilisation d'un *Rendez-Vous* (Figure 1.5). Plutôt que d'envoyer directement un message volumineux (dont le récepteur ne saurait pas forcément quoi faire à la réception), l'émetteur envoie une *annonce* (demande de rendez-vous) de petite taille au récepteur. Ce dernier peut ainsi prévoir un emplacement mémoire destination où transférer directement les données à leur réception. Cette étape effectuée, il envoie un message d'acquittement à l'émetteur signifiant qu'il est prêt à la réception du message. L'acquittement reçu, l'émetteur effectue alors l'envoi réel des données. Ce principe évite

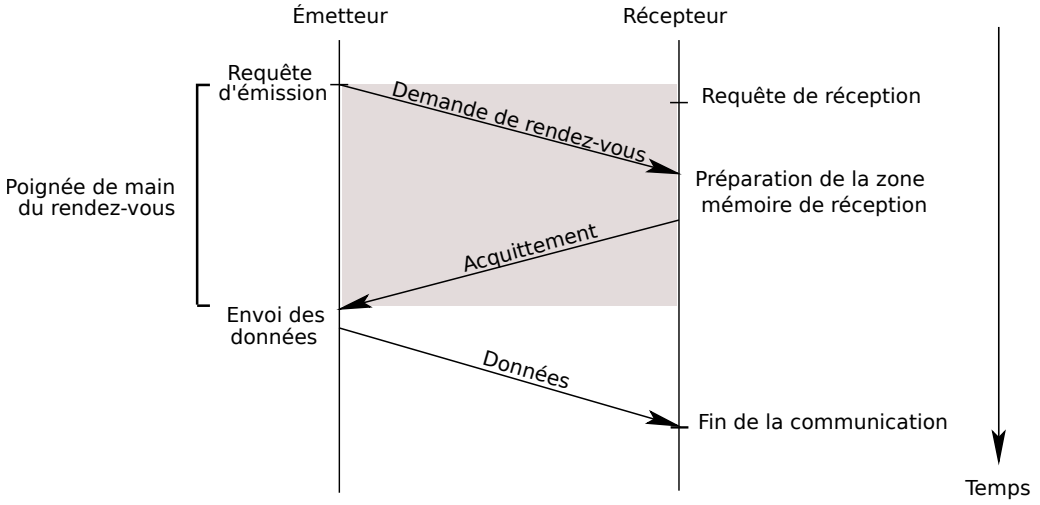


FIGURE 1.5 – *Protocole de Rendez-Vous.*

l'utilisation d'un tampon intermédiaire côté récepteur, mais permet également d'éluder les copies supplémentaires en émission. Les en-têtes peuvent ainsi être transmises au cours de la *poignée de main* du Rendez-Vous. Dans le cas de petits messages, le temps de l'échange préalable au transfert est plus grand que les recopies et un tel procédé est dénué de sens. Un seuil de rendez-vous, idéalement égal à la taille de message à partir de laquelle cette méthode devient rentable, est utilisé pour définir la stratégie à employer, avec ou sans Rendez-Vous.

Un autre facteur important dans la conception des réseaux rapides concerne la notification de terminaison des requêtes d'émission et de réception. Celle-ci peut être faite par *scrutation* (attente active), assurant une très bonne réactivité, mais consommatrice de temps processeur au détriment des calculs de l'application. Au contraire, les périphériques peuvent signaler un événement réseau à l'aide d'une *interruption*. Cette opération coûteuse (plusieurs microsecondes) laisse le processeur libre pour le calcul, mais pénalise grandement la latence de bout en bout espérée sur réseau rapide. Le traitement des événements réseau est donc basé sur une composition intelligente entre scruta-

tion (pour les événements arrivant tôt) et interruption (pour les événements tardifs) pour assurer un bon compromis entre réactivité et consommation processeur.

Suppression du système d'exploitation sur le chemin critique
Outre la réduction de la pile protocole et l'optimisation des transferts bruts, un gain notable de latence peut être obtenu en contournant le système d'exploitation. Lors de l'envoi d'un message sur le réseau, l'accès au périphérique est normalement à la charge du système d'exploitation. Sur les machines modernes, le coût d'un appel système est de l'ordre de quelques centaines de nanosecondes et peut constituer un pourcentage notable de la latence du transfert d'un faible volume de données. Une innovation apportée à l'emploi de réseaux rapides consiste à éliminer l'intervention du système d'exploitation (*OS-bypass*) et permettre à l'application d'effectuer le transfert depuis l'espace utilisateur. Pour ce faire, l'interface de communication s'appuie sur une extension du noyau supportant la projection de la mémoire de la carte dans l'espace d'adressage du processus, accomplie au cours de l'initialisation.

1.1.3 Description des principaux réseaux haute performance

Depuis le développement des premières technologies de réseaux rapides et l'installation des grappes dans l'arène du calcul haute performance, les constructeurs n'ont cessé de faire évoluer les technologies proposées. Ce paragraphe présente les technologies les plus représentatives qui ont dominé le paysage des grappes de calcul ces dernières années, et les performances actuelles qu'elles exposent.

MYRINET et MYRI-10G
Grâce à l'introduction en 1995 du réseau MYRINET [52], la société MYRICOM [51] a pendant longtemps été un leader dans le domaine des réseaux haute performance. Cette technologie est dotée de cartes performantes et faciles à programmer dont les premières spécifications étaient libres. Elle a ainsi fait l'objet de nombreux travaux académiques qui ont contribué à l'évolution des protocoles réseau, notamment VMMC (*Virtual Memory-Mapped Communication* [55]), FM (*Fast Messages* [56]) et BIP (*Basic Interface for Parallelism* [57]). La dernière génération de cartes MYRINET, nommée MYRI-10G, a été introduite en 2005. Elle est dotée d'un processeur RISC (le LANai) cadencé à 333 MHz et possède 2 Mo de SRAM et un moteur DMA. Elle est exploitable par l'interface de communication MX (*Myrinet eXpress* [54]) qui a été conçue grâce à l'expertise accumulée au fil des ans. Sa sémantique s'accorde avec celle du standard MPI utilisé pour la communication sur grappe. Elle permet à cette technologie d'exhiber par exemple une latence de 2,1 μs et un débit de 1223 Mo/s [3] entre deux INTEL XEON X5460 cadencés à 3,16 GHz.

QSNET II
En 2003, la société QUADRICS a lancé sa dernière génération de cartes, QSNET II [46], utilisée avec l'interface de communication ELAN4. Grâce à des transferts PIO particulièrement efficaces portés par la puissance du processeur intégré, cette technologie offrait une latence record de l'ordre

3. 1 Mo = 1024^2 octets

d'une microseconde. Sa bande passante était cependant limitée à 900 Mo/s par la fréquence du bus mémoire. L'année de sa sortie, cette technologie équipait 6 clusters classés parmi les 10 premiers calculateurs du Top500 [5]. Si la latence était excellente, le prix de l'installation très élevé puis l'absence de support des bus PCIE ont freiné la diffusion de QSNET II. L'entreprise QUADRICS a fermé en 2009 avant la sortie de ELAN4/PCIE.

INFINIBAND

La famille de réseaux INFINIBAND est née d'un consortium de constructeurs (IBM, SUN, INTEL, HEWLETT-PACKARD...) dans les années 90. L'idée originale était de définir un standard d'architecture des entrées-sorties à venir : remplacer le bus PCI et les systèmes d'accès au système de stockage et au réseau [48]. Après le retrait du constructeur INTEL et l'annonce du bus PCIE pour remplacer le bus PCI, le projet s'est concentré vers les réseaux haute performance.

Cette technologie est basée sur la notion de RDMA (*Remote Direct Memory Access*) qui implique un accès direct à la mémoire des nœuds distants. INFINIBAND est en effet doté de cartes capables de décider où lire (*RDMA read*) ou écrire (*RDMA write*) les données dans la mémoire, plutôt que de passer par un tampon dédié à la carte. Elles s'appuient sur un enregistrement des zones mémoire au cours de l'initialisation et permettent des transferts zéro-copie performants. L'interface de communication la plus employée est l'interface VERBS, bien que son utilisation soit assez compliquée car très bas niveau. Les réseaux INFINIBAND offrent actuellement d'excellentes performances avec une latence de l'ordre d'une microseconde et des débits qui dépassent 3 Go/s avec des liens QDR (*Quad Data Rate*). Ces réseaux rapides sont ainsi devenus les plus populaires du HPC et ne cessent de prendre des parts de marché, notamment au détriment des réseaux Myrinet, détrônés de leur position dominante depuis 2006 (Figure 1.6). En 2011, plus de 40% des grappes du Top500 sont équipées de la technologie INFINIBAND.

Ethernet

L'universalité des cartes Ethernet dans les ordinateurs de bureau a affirmé la position dominante de ce réseau et les efforts développés par les constructeurs de réseaux rapides n'ont pas réussi à évincer cette technologie des grappes de calcul. Depuis une dizaine d'années, le nombre de clusters dotés de réseaux Ethernet dont les évolutions matérielles ont apporté une amélioration significative de débit, ne cessent d'augmenter. Après FastEthernet et GigabitEthernet, la production des cartes 10 Gigabit Ethernet, assure la place de cette technologie dans les grappes de calcul grâce à leur rapport performance/prix, et près de 45% des machines du Top500 en sont équipées (Figure 1.6). Cette technologie bénéficie désormais d'un grand nombre des mécanismes déployés dans les réseaux rapides. De plus, le coût de la pile TCP/IP, principal inconvénient d'Ethernet pour les contraintes du HPC, peut être évité grâce à l'utilisation de protocoles alternatifs. Le projet GAMMA (*Genoa Active Message MAchine* [114]) a amorcé ces travaux et permet une réduction notable de la latence au-dessus d'Ethernet. Il nécessite cependant une modification du pilote Ethernet et reste limité à certains chipsets INTEL. Des travaux récents, notamment les projets OPEN-MX et ROCEE, proposent des solutions portables et très prometteuses. OPEN-MX [116] repose sur une implémentation de la pile protocole MX de Myrinet sur les différentes interfaces Ethernet. Son utilisation requiert un noyau récent (2.6.15 ou plus) mais ne demande pas de modification de pilotes. Les performances sur Ethernet 10-Gigabit atteignent 6 μs de latence et frôlent

la bande passante théorique. RoCEE [118] pour *RDMA over Converged Enhanced Ethernet* associe le RDMA d'INFINIBAND avec le nouveau standard de protocole Ethernet sans perte, CEE, permettant des accès directs en mémoire entre les nœuds de la grappe. Il diminue ainsi la consommation de ressources et abaisse les temps de latence jusqu'à 1,3μs pour Ethernet 10-Gigabit. On assiste ainsi à une convergence entre les réseaux rapides et traditionnels.

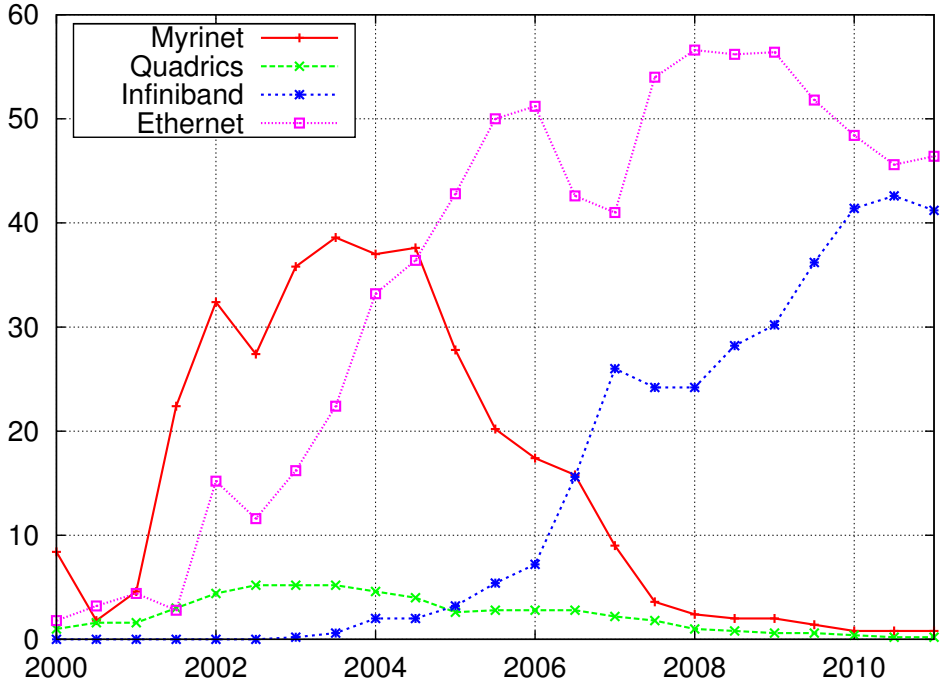


FIGURE 1.6 – *Évolution du pourcentage de présence des différentes familles de technologies réseau sur les machines du Top500 ces 10 dernières années.*

1.1.4 Bilan

Les grappes se sont largement implantées dans le domaine du calcul intensif grâce à leur excellent rapport performance/prix et à l'émergence de technologies réseau efficaces qui réduisent le coût des communications inter-nœuds. Les constructeurs de ces technologies ont sans cesse optimisé leurs produits, axant leurs efforts sur le développement du matériel mais aussi sur la conception de supports logiciels adaptés. La dernière décennie a ainsi vu mettre au point des réseaux offrant des performances spectaculaires avec des débits de plusieurs Go/s, et des latences de l'ordre d'une microseconde, fondées sur l'utilisation de matériel et de mécanismes de transferts toujours plus performants. Aujourd'hui, alors que la technologie standard Ethernet affiche des performances théoriques proches de celles des technologies réseau spécialisées, on assiste à une convergence entre réseaux rapides et traditionnels grâce aux progrès mis en œuvre dans les interfaces de communications.

Si l'évolution des technologies réseau a permis l'insertion des grappes de calcul dans le do-

maine du HPC, la puissance interne à chaque nœud de la grappe joue un rôle prépondérant dans la puissance de calcul globale. En effet, si les communications peuvent être un facteur limitant dans l'exécution d'une application parallèle, la puissance de calcul brute reste un facteur majeur d'accélération. Il serait de plus problématique que les nœuds ne soient pas suffisamment performants pour d'une part traiter les données reçues lors des communications et d'autre part effectuer les calculs intrinsèques au déroulement de l'application. La puissance des grappes a ainsi été enrichie grâce à l'utilisation de nœuds de calcul de plus en plus puissants, mais aussi de plus en plus complexes.

1.2 Évolution des nœuds de calcul au sein des grappes

Bien que les grappes de calcul s'appuient sur du matériel standard, les ordinateurs utilisés sont généralement des modèles haut de gamme en termes de puissance et de performance. Les premières grappes de calcul historiques étaient constituées de simples stations de travail monoprocesseurs, mais les grappes ont très vite intégré des architectures composées de plusieurs processeurs. Ces machines multiprocesseurs offraient ainsi une augmentation notable de la puissance de calcul tout en conservant un rapport performance/prix intéressant. Par exemple, la grappe du LLNL[4] construite par LINUX NETWORX et QUADRICS et classée en cinquième place du Top500 en 2002, était composée de 1152 machines bi-processeurs INTEL XEON cadencés à 2.4 GHz.

Ces architectures, dites SMPs (*Symmetric MultiProcessing*) intègrent plusieurs processeurs connectés à la mémoire via un même bus comme illustré sur la figure 1.7. Elles augmentent ainsi le nombre de ressources de calcul par nœud, mais insèrent un second niveau de parallélisme au sein des grappes, qui disposent ainsi de parallélisme externe et interne. Sur ces architectures, les accès mémoire effectués par les différents processeurs peuvent être source de contention au niveau du bus qui devient alors un véritable goulot d'étranglement. Les machines SMP possèdent ainsi un nombre restreint de processeurs, le plus souvent deux ou quatre et rarement plus de seize.

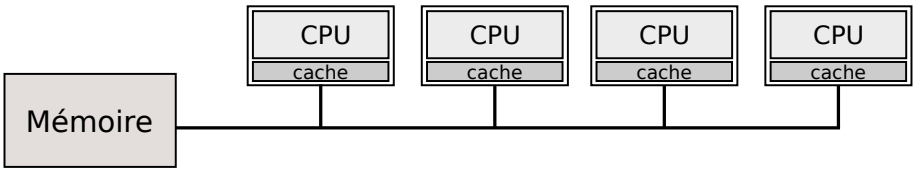


FIGURE 1.7 – *Architecture SMP.*

Le calcul haute performance a également profité des innovations apportées au sein des processeurs par les fondeurs, offrant toujours plus de puissance grâce à l'augmentation de la fréquence. La technologie se heurte aujourd'hui à une barrière thermique qui limite cette fréquence, contraignant les constructeurs à déployer de nouvelles méthodes pour accroître l'efficacité des ordinateurs.

Alors que les progrès de miniaturisation libèrent de l'espace sur les puces, la solution choisie par les constructeurs s'est orientée vers le parallélisme. Avec l'ajout ou la duplication de composants, les processeurs ont ainsi gagné en performance mais aussi en complexité. Le développement et la

4. Lawrence Livermore National Laboratory

recherche dans ce domaine ont abouti à la généralisation du parallélisme interne aux processeurs dont les dernières générations équipant les PCs de bureau disposent ainsi de plusieurs "*cœurs* ".

1.2.1 La révolution du multicœur

Fréquence et miniaturisation : des machines monoprocesseurs au multicœur
Les conjectures de Moore annonçaient une multiplication par deux de la densité des transistors sur les microprocesseurs tous les deux ans. Grâce à l'amélioration de la finesse de gravure, cette prédiction s'est révélée exacte pendant de nombreuses années et s'est traduite par une augmentation de la fréquence et de la puissance des processeurs. Alors que celle-ci semblait compromise par les problèmes de dissipation thermique qui plafonnent les fréquences des processeurs autour de 4 GHz, les progrès de miniaturisation ont relancé l'évolution des processeurs et de leurs performances. En une trentaine d'années, nous sommes passés de microprocesseurs de quelques milliers de transistors, 29000 par exemple pour le 8086 d'INTEL (1979), à plusieurs milliards de transistors avec 2,3 milliards pour le Nehalem-EX Xeon octo-cœur sorti en mars 2010.

Dans un premier temps, le gain de place a permis aux constructeurs d'ajouter des composants à leurs processeurs (registres, pipelines, prédiction de branchement, réordonnancement d'instructions, etc) produisant des processeurs de plus en plus riches et sophistiqués. Parmi les évolutions marquantes, on pourra noter la naissance des processeurs *superscalaires* qui permettent l'exécution simultanée de plusieurs instructions d'un programme séquentiel lorsque la dépendance des données le permet, chacune dans un pipeline différent.

Dans la recherche au perfectionnement, les constructeurs ont mis en place des techniques telles que le *Simultaneous MultiThreading* (SMT, appelé *HyperThreading* chez INTEL), pour alimenter aux mieux les multiples unités fonctionnelles. Cette technologie autorise l'utilisation concurrente d'un pipeline par plusieurs flots d'exécution (*threads*). Elle offre un premier niveau de parallélisme perçu comme des *processeurs virtuels* (processeurs logiques). En pratique, si le gain de performances apparaît notable dans certains cas, il semble discutable dans d'autres [45]. Une détérioration est même possible, par exemple lorsque les threads concurrents utilisent le même type d'instructions (flottantes, entières,...). Ces résultats incertains poussent généralement les scientifiques à désactiver ce mécanisme pour le calcul intensif.

Aujourd'hui, plutôt que de complexifier davantage les processeurs déjà très sophistiqués, la miniaturisation permet de graver directement plusieurs processeurs sur une même puce (Figure 1.9), on parle alors de processeurs *multicœurs.* Les puces multicœurs sont la voie de développement choisie par les fondeurs et constituent le noyau des architectures actuelles.

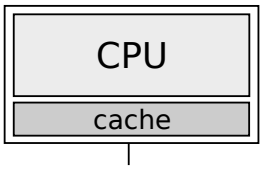


FIGURE 1.8 – *Processeur sans cœur.*

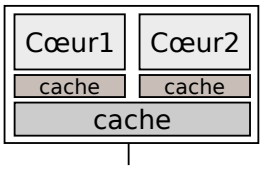


FIGURE 1.9 – *Puce bi-cœur.*

Une organisation hiérarchique

Tous les grands constructeurs proposent désormais des déclinaisons multicœurs de leur processeurs. Ces dernières années ont marqué une diffusion des puces bi-cœurs et quadri-cœurs, qui sont devenues le standard du marché grand public. Les processeurs *core i*, dernière génération de puces proposées par le constructeur INTEL, sont par exemple composées de 2 ou 4 cœurs hyperthreadés, (voire même 6 cœurs hyperthreadés pour le *core i7-980X*). La tendance du multicœur se retrouve même dans les consoles de jeu comme en témoignent le succès de la PLAYSTATION 3, équipée d'un processeur multicœur hétérogène, le Cell B.E., ou de la XBOX 360 qui dispose d'un processeur Xenon à trois cœurs produit par IBM.

Les fondeurs offrent également pour les centres de calcul des variantes haut de gamme à 6, 8 où 12 cœurs, tel que le processeur *Magny-Cours* [59] proposé par AMD depuis mars 2010, voire bientôt 16 cœurs avec le processeur *Interlagos* annoncé pour 2011.

Les multicœurs ont la particularité de regrouper plusieurs processeurs sur une même puce, produisant ainsi des machines multiprocesseurs à moindre coût. De la même façon que les machines SMPs, chaque cœur accède à la mémoire au travers d'un bus ou réseau d'interconnexion. L'organisation des puces multicœurs n'est cependant pas assimilable à celle de ces architectures plates et varie par la présence de ressources partagées entre les cœurs, notamment les zones de mémoire cache. En effet, la multiplication des composants au sein des processeurs a introduit la duplication des *caches*. Différents niveaux de cache, de tailles et de vitesses variées sont souvent juxtaposés sur le processeur pour répondre aux besoins hétéroclites des applications. Sur les puces multicœurs, certains niveaux de cache peuvent être partagés entre plusieurs des unités de calcul comme illustré par la Figure 1.9.

Il en résulte ainsi une hiérarchie de cache dont l'organisation varie selon les modèles et les constructeurs. La Figure 1.10 présente la structure d'une puce 6 cœurs INTEL Xeon Dunnington. Sur cet exemple chaque cœur dispose de son propre cache L1. Un cache L3 est commun à l'ensemble des cœurs tandis que les caches L2 sont associés à des paires de cœurs.

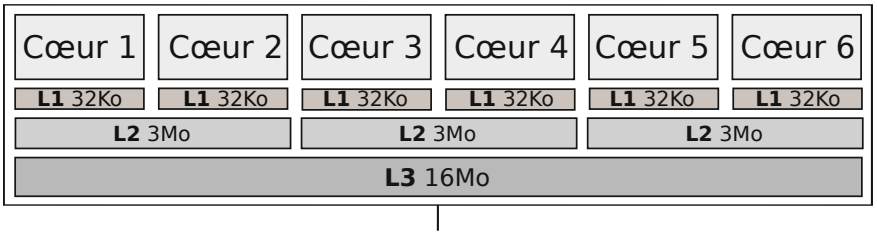


FIGURE 1.10 – *Puce hexa-cœur* INTEL *Xeon Dunnington X7460.*

L'agencement des différents niveaux de cache est responsable d'affinités entre les cœurs. Et effet, un cache commun à deux cœurs permet un partage de données entre les processus qui s'exécutent sur ces cœurs. Tant que les données tiennent dans le cache, les performances peuvent en être considérablement améliorées (Section 2.3). Au contraire, une utilisation concurrente du cache peut avoir un effet dégradant sur les performances, chaque processus ne disposant concrètement que de la moitié de celui-ci. Combiné avec le partage de la bande passante du bus en dehors de la puce, les performances des machines multicœurs subissent un fort impact des accès concurrents.

Avec la diffusion des puces multicœurs, les grappes de calcul sont le plus souvent articulées autour

de machines multiprocesseurs-multicœurs. Les processeurs des machines SMP sont ainsi remplacés par des puces multicœurs comme illustré en Figure1.11. Le modèle simple et uniforme des architectures SMPs est ainsi complexifié par la hiérarchie de cache intégrée au sein des puces, qu'il devient impossible d'ignorer dans la recherche de performance. De plus, le problème de passage à l'échelle de ces architectures est amplifié par l'augmentation du nombre de cœurs qui génèrent d'autant plus d'accès concurrents sur le bus mémoire.

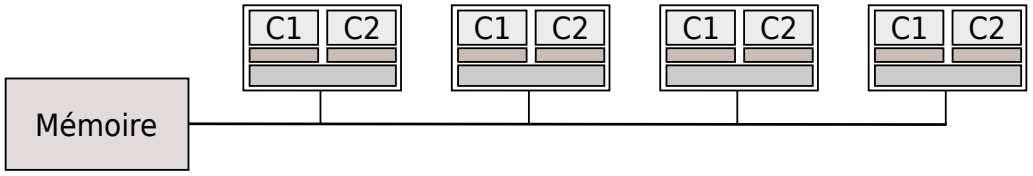


FIGURE 1.11 – *Architecture SMP multicœurs.*

1.2.2 Le retour du NUMA

Pour créer des machines parallèles de grande taille, il est indispensable de pallier la congestion suscitée par l'accès à la mémoire via un unique bus. Une solution courante, développée dans les années 90, consiste à distribuer la mémoire en différents *bancs mémoire*. Il en résulte des architectures multiprocesseurs à mémoire partagée, composées de plusieurs ensembles "banc mémoire - processeurs" appelés *nœuds* et reliés au travers d'un commutateur ou d'un réseau d'interconnexion (Figure 1.12).

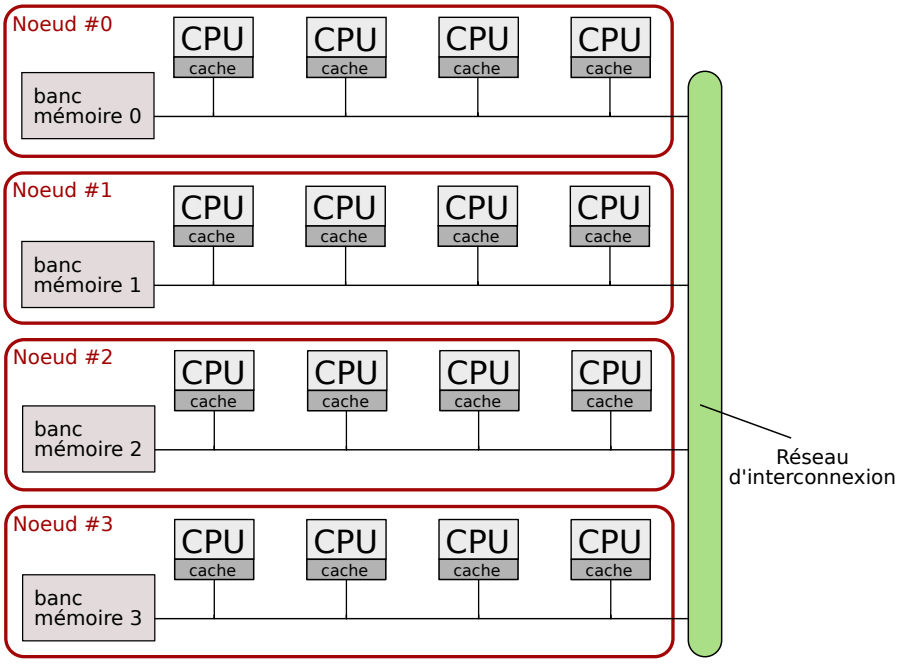


FIGURE 1.12 – *Architecture NUMA multicœur à quatre nœuds.*

De telles machines sont dites à accès mémoire non uniformes (*Non Uniform Memory Access*). Les temps d'accès mémoire dépendent de la position relative du processeur et de la mémoire accédée. La latence d'accès à la mémoire *locale* (sur le même nœud que le processeur) est plus faible que celle d'un accès à la mémoire *distante* fait au travers du réseau d'interconnexion. Cette non-uniformité est quantifiée par un *facteur* NUMA (voire plusieurs dans le cas de machines à topologie complexe ou hiérarchique). Ce facteur équivaut au rapport entre le temps d'accès à la mémoire distante et celui à la mémoire locale, et varie fortement selon les architectures.

La complexification des architectures et l'entrée en scène des puces multicœurs, a suscité un regain d'intérêt pour ces structures qui se multiplient dans le domaine du calcul haute performance grâce à la création de nouvelles technologies d'interconnexion.

De nouveaux réseaux d'interconnexion

Une méthode classique pour concevoir une architecture NUMA consiste à assembler plusieurs architectures de type SMP autour d'un réseau interne d'interconnexion. Un grand nombre de serveurs ont été bâtis sur ce modèle. C'est le cas des serveurs basés sur les processeurs ITANIUM d'INTEL très présents il y a quelques années, tels que les machines BULL NOVASCALE (assemblage de plusieurs QBB (*Quad Building Block*) comprenant chacun de 2 ou 4 processeurs), ou les serveurs ALTIX de la société SGI regroupant jusqu'à plusieurs centaines d'Itanium. Le facteur NUMA de ces architectures variait généralement entre 1 et 3.

Par opposition à ces architectures régulières, sont apparues il y a quelques années de nouvelles architectures NUMA grâce au développement de nouveaux systèmes d'interconnexion. L'assemblage de plusieurs bus mémoire par un réseau d'interconnexion dédié étant onéreux, AMD a développé le système HYPERTRANSPORT [39, 40], souvent appelé *bus* mais qui est en fait un réseau d'interconnexion.

Plutôt que d'être connecté à un bus mémoire centralisé, les processeurs AMD OPTERON [41] sont connectés à plusieurs *liens* HYPERTRANSPORT (1 à 4 suivant les modèles). Chaque processeur est doté d'un contrôleur mémoire et possède son propre banc mémoire qui lui est directement connecté par un lien HYPERTRANSPORT (Figure 1.13), faisant de chaque ensemble "banc mémoire/processeur" un nœud NUMA. Les machines multiprocesseurs OPTERON sont ainsi caractérisées par des connexions "point-à-point" au travers des liens d'interconnexion. Le temps d'accès aux nœuds NUMA, ou aux périphériques d'entrées-sorties (eux aussi connectés à un lien), est déterminé par le nombre de liens traversés, et on observe des facteurs NUMA variant entre 1 et 2 [42].

Ce système de connexion a permis aux processeurs OPTERON de se démarquer des processeurs concurrents. Il offrait en effet une bande passante de très loin supérieure à celle des bus mémoire utilisés avec des processeurs INTEL, ainsi qu'une latence inégalée jusqu'à la sortie des *Core2* en 2006. Ces architectures ont ainsi connu un large succès dans le monde du calcul scientifique, représentant jusqu'à 22% des systèmes du Top500 [5] en 2007.

La réponse d'INTEL face à cette technologie est le système d'interconnexion QUICKPATH INTERCONNECT (QPI), qui remplace désormais le bus mémoire externe bidirectionnel (*Front Side Bus*) dans ses nouvelles architectures (Nehalem [43], Tukwila & brothers). On retrouve un schéma équivalent au système d'AMD : chaque processeur dispose d'un contrôleur mémoire intégré et

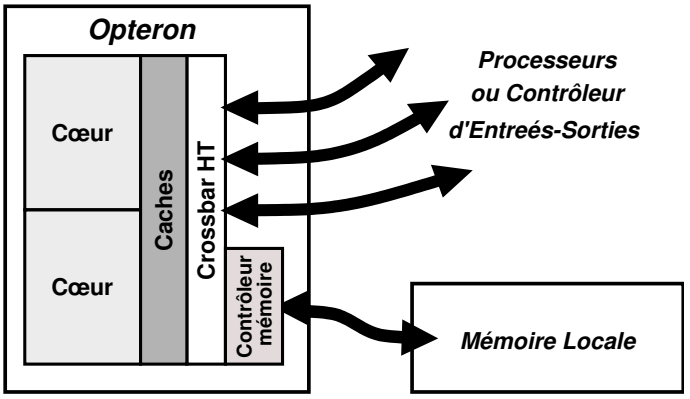


FIGURE 1.13 – *Processeur Opteron.*

se trouve relié à un banc mémoire local, à d'autres processeurs, ou à des périphériques d'entrées-sorties au travers d'un lien d'interconnexion. Le débit record annoncé pour ce système d'interconnexion, 25,6 Gbit/s (6.4 GigaTransferts/s par lien) [44], pour une fréquence de 3.2GHz, et la popularité des derniers processeurs INTEL ont replacé ce constructeur comme leader du marché du HPC. La technologie QPI est ainsi intégrée dans 64,4% des machines du TOP500 contre 13,8% pour AMD HYPERTRANSPORT. La version 3.1 de la technologie HYPERTRANSPORT, parue quelques mois après la sortie QPI expose cependant des performances comparables à celle-ci, avec 25,6 Gbit/s (6.4 GigaTransferts/s par lien) pour 3.2GHz de fréquence.

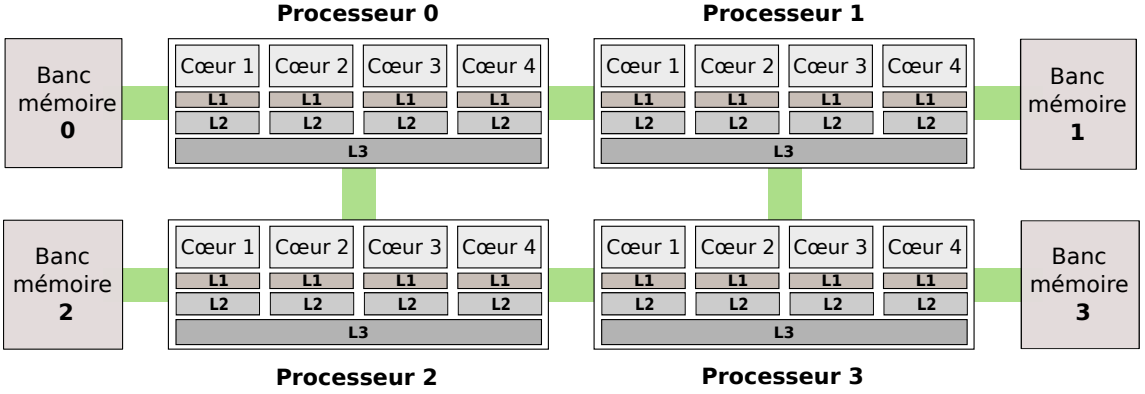


FIGURE 1.14 – *Quadri-processeur AMD Opteron quadri-cœur Barcelona 8347HE.*

Grâce à ces technologies, les architectures NUMA multicœurs sont devenues très populaires au sein des grappes de calcul. La Figure 1.14 illustre la structure hiérarchique complexe que peut avoir un nœud calcul (ici un quadri-processeur quadri-cœur AMD OPTERON de notre plateforme de test). Cette architecture n'est bien sûr qu'un exemple parmi d'autres et les organisations sont propres aux différentes plateformes proposées par les constructeurs. Elles présentent des hiérarchies de caches variées, une ou plusieurs sockets [5] par nœud NUMA, (voire même plusieurs nœuds NUMA par socket pour les derniers processeurs OPTERON), disposent ou non d'hyperthreading, emploient différentes stratégies de numérotation des cœurs, etc. La conséquence directe de

5. "Puces physiques" pouvant être juxtaposées sur le processeur.

la complexification topologique et d'une telle variété d'organisation est une difficulté croissante à exploiter proprement ces machines contemporaines, marquées par d'importantes contraintes de localité. En effet, comme nous le verrons en Section 2.3, la forte structure hiérarchique interne à chaque nœud a un impact crucial sur les performances des applications.

1.2.3 Tendances : une complexification grandissante

Il y a quelques années on entendait parler de futures machines à plusieurs centaines de cœurs [12]. Aujourd'hui, les problématiques de passage à l'échelle et de hiérarchisation mises à jour avec la diffusion du multicœurs ont recadré ces prévisions. La tendance actuelle reste toutefois à l'intégration de composants au sein des puces. Alors que les progrès de miniaturisation ont permis aux fondeurs de perfectionner leurs processeurs jusqu'aux limites de la sophistication, puis de multiplier les cœurs, l'espace libéré permet aujourd'hui d'explorer le potentiel d'intégration de composants externes au sein des processeurs.

Les plateformes CENTRINO d'Intel destinées aux ordinateurs portables offraient déjà une juxtaposition de composants sur une même puce pour réduire la consommation électrique. En pratique, la véritable intégration logique a commencé avec l'ajout du contrôleur mémoire HYPERTRANSPORT au sein des processeurs OPTERON, pour multiplier les performances mémoire. Le contrôleur QPI a ensuite été intégré dans les Nehalem sur le même principe. Après le contrôleur mémoire, on assiste à l'intégration des contrôleurs d'entrées-sorties annoncée pour les nouvelles générations de processeurs INTEL (*Sandy-Bridge*) et AMD (*Bulldozer*).

En parallèle, la course à la performance et l'introduction de problématiques de consommation électrique (*Green Computing*) a engendré l'explosion de la recherche concernant l'utilisation de processeurs graphiques (GPU) ou de processeurs hétérogènes tels que le Cell. L'utilisation de plateformes hétérogènes combinant des GPUs et des CPUs s'est ainsi installée dans le domaine du HPC [5]. Aujourd'hui ces tendances se rejoignent avec l'annonce du processeur AMD Fusion qui intégre des GPUs dans le processeur [7].

Les générations futures de processeurs s'annoncent ainsi marquées par une hétérogénéité et une complexification grandissante. Les spéculations sur les plateformes à venir mentionnent la généralisation de l'hétérogénéité et l'abandon de cohérence de cache. De tels changements augurent ainsi de nouveaux niveaux de hiérarchie, et de véritables défis pour les programmeurs qui devront adapter les modèles de programmation à ces structures.

1.3 Bilan : un parallélisme hiérarchique

Les révolutions technologiques dans le domaine de l'informatique ont conduit à une transformation phénoménale de l'architecture des calculateurs. La création de technologies réseau performantes a mené à l'émergence des grappes de calcul et l'utilisation de nœuds de calcul multiprocesseurs s'est généralisée.

La puissance des processeurs s'est longtemps vue augmentée par la multiplication des transistors et de la fréquence, mais les problèmes de dissipation thermique devenus inévitables, les fondeurs

se sont alors tournés vers une augmentation du parallélisme avec l'intégration d'unités de calcul supplémentaires, donnant naissance aux puces multicœurs. Les processeurs multicœurs, standard des architectures modernes, ont introduit un nouveau degré de complexité au sein des machines et des obstacles de scalabilité. Le développement des technologies HYPERTRANSPORT et QPI offrent la possibilité d'assembler un large nombre de processeurs multicœurs autour d'un réseau d'interconnexion mais entraîne le retour d'architectures à accès mémoire non uniforme.

Les topologies des nœuds interconnectés en cluster sont ainsi à l'image de véritables poupées russes, composées de multiniveaux de parallélisme, combinant différentes technologies (machines multiprocesseurs multicœurs NUMA par exemple), et dont l'organisation varie fortement selon les constructeurs et les plateformes. La hiérarchie intrinsèque aux machines est connue pour avoir un impact conséquent sur les performances des applications, mais l'imbrication des structures est devenue telle que ces effets sont difficilement prévisibles. Alors qu'en 10 ans la puissance de calcul des calculateurs a été multipliée par 1000 [5], l'exploitation des structures complexes résultantes est devenue un véritable casse-tête pour les programmeurs et le fossé entre les performances théoriques et les performances soutenues se creuse. De plus, l'inclinaison à l'intégration de composants divers qui se dessine pour les architectures futures présage une amplification de ces difficultés, et la diffusion de machines non seulement hiérarchiques mais aussi hétérogènes.

Exploiter les machines complexes est donc devenu un problème crucial et délicat qui fait l'objet de nombreuses recherches. Celles-ci s'articulent autour de la compréhension des effets de la topologie des machines contemporaines et la projection des applications, construites sur des modèles de programmation antérieurs à cette hiérarchisation, sur ces architectures modernes.

Chapitre 2

Exploiter les architectures parallèles

Sommaire

2.1 Modèles de programmation pour les architectures parallèles 26
2.1.1 Programmation par mémoire partagée 26
2.1.2 Programmation pour les systèmes à mémoire distribuée 28
2.1.3 Mémoire virtuellement partagée 30
2.1.4 Modèles de programmation hybrides 31
2.1.5 Bilan 32
2.2 Gestion des communications dans les bibliothèques MPI 32
2.2.1 Stratégies de transferts utilisées par les bibliothèques MPI 33
2.2.2 Opérations collectives et réduction du trafic interprocessus 36
2.2.3 Affiner les algorithmes de communication 37
2.2.4 Bilan 38
2.3 Impact des architectures contemporaines 39
2.3.1 Quels défis pour les programmeurs ? 39
2.3.2 Quels supports pour détecter et exploiter la topologie des architectures ? 42
2.3.3 Des conséquences de la topologie sur les communications réseau 46
2.4 Bilan 48

Les machines parallèles sont mises à contribution dans la résolution de calculs scientifiques de grande taille, qui doivent être exécutés dans un laps de temps suffisamment court et avec un degré d'exactitude et un niveau de détail toujours croissant. De nombreux modèles de programmation parallèle ont été construits pour utiliser simultanément les différentes unités de calcul disponibles et accélérer l'exécution des applications. Ils répondent au besoin de partage du travail en différentes tâches et à la gestion de leur interdépendance qui nécessite des échanges de données et des synchronisations pour assurer la cohérence et le bon déroulement des calculs.

Ce chapitre présente les modèles de programmation et les standards les plus utilisés au sein des machines de calcul parallèles. Les différents niveaux de parallélisme introduits au cours de l'évolution de l'architecture des calculateurs (Section 1.2) ajoutent de fortes contraintes de localité qui accroissent les difficultés de programmation. Les scientifiques à l'origine des applications parallèles ne sont pas tous informaticiens et ne possèdent pas toujours les connaissances nécessaires pour

tirer de ces modèles les meilleures performances sur les architectures modernes. De plus, les programmes sont généralement créés avec des objectifs de pérennité et sont susceptibles d'être exécutés sur différentes architectures matérielles. Il est donc nécessaire d'assurer une certaine facilité de programmation et la portabilité des applications.

2.1 Modèles de programmation pour les architectures parallèles

Le calcul scientifique présente des besoins spécifiques avec de très grandes quantités de données à traiter, pour lesquelles les traitements peuvent être répartis sur les multiples nœuds de calcul d'une grappe. Les applications parallèles en charge de ces calculs doivent être capables de distribuer le travail entre les différentes unités de calcul afin de résoudre les calculs en un temps acceptable. L'interdépendance entre les tâches soumet les processus participant au travail à des synchronisations et des échanges de données. Le calcul parallèle est ainsi caractérisé par des communications intensives. Ces besoins ont induit la mise au point de modèles de programmation spécifiques et favorisé l'émergence d'interfaces de programmation dédiées. Les grappes de calcul auxquelles nous nous intéressons font intervenir deux types d'échanges : les échanges *intra-nœud* effectués au sein même d'un nœud de calcul, et les transferts *inter-nœuds* intervenant entre des nœuds distincts au travers de réseaux rapides.

2.1.1 Programmation par mémoire partagée

Dans le cadre de la programmation parallèle, l'utilisation de la mémoire partagée repose sur l'exploitation d'un espace d'adressage unique (mémoire virtuelle) et commun aux tâches concurrentes. Ce système propose des échanges implicites et efficaces entre les tâches en utilisant la mémoire de la machine comme *zone de transfert* dans laquelle les données sont déposées ou lues. Le coût de communication s'apparente ainsi à ceux des temps d'accès mémoire, néanmoins, pour assurer la consistance et la cohérence des données, l'emploi de méthodes de synchronisation et de verrouillage (synchronisation explicite, utilisation d'instructions atomiques, exclusion mutuelle, attente sur condition, ect.) est indispensable. Les solutions basées sur ce modèle s'appuient ainsi sur l'utilisation d'interfaces ou de langages de programmation dédiés permettant d'exprimer la manière dont les différents traitements nécessaires à la réalisation des calculs doivent être créés, s'exécuter et se synchroniser.

2.1.1.1 Multithreading

Une première méthode pour créer des applications parallèles en mémoire partagée consiste à exprimer le parallélisme *à la main*, à l'aide de bibliothèques de processus légers (*threads*) qui fournissent des primitives de création, gestion et destruction de threads.

Historiquement, différents constructeurs proposaient leurs propres bibliothèques de threads. Pour des raisons de portabilité, l'interface standard de Threads POSIX (PTHREAD) a été développée pour les systèmes UNIX. De très bas niveau, celle-ci offre une expressivité extrêmement fine du parallélisme et permet de mettre en œuvre de nombreuses optimisations. En pratique, ce mode

de programmation est complexe. Le programmeur doit être capable d'exprimer la granularité appropriée au programme, d'établir les synchronisations et verrouillages nécessaires, et d'assurer la portabilité de ses optimisations. Il requiert de ce fait un niveau de programmation technique élevé.

2.1.1.2 Langages de programmation

Pour faciliter l'écriture de programmes parallèles, les programmeurs peuvent se tourner vers des langages de programmation spécifiques au modèle de programmation par mémoire partagée. Ceux-ci offrent des approches moins flexibles que l'utilisation de bibliothèques de threads, mais simplifient le travail des développeurs auxquels un certain nombre de détails techniques sont cachés.

OPENMP

Développé par un groupe d'industriels et de spécialistes en compilation depuis 1997, le standard OPENMP [109] (*Open Multi-Processing*) désigne une extension de langage (C, C++ et Fortran). Il s'appuie sur un parallélisme de boucle *fork-join* exprimé à l'aide de directives (*pragmas*). Celles-ci sont interprétées par le compilateur pour générer automatiquement le parallélisme et la gestion des threads sous-jacente (création, destruction, synchronisation, etc.). OPENMP permet une parallélisation incrémentale du code existant simple à mettre en œuvre. Les directives peuvent être ignorées pour rebasculer en mode d'exécution séquentiel, ou enrichies d'indications (mots-clés) définissant par exemple le nombre de threads à créer, la distribution des tâches entre les threads, etc. La faible granularité du parallélisme généré convient plus ou moins bien aux applications, et l'utilisation automatique de la mémoire partagée souffre d'un véritable problème de passage à l'échelle, principale faiblesse d'OPENMP. En effet, s'il s'avère efficace sur les machines SMPs, OPENMP ne tient pas encore compte de la localité mémoire prescrite pour les architectures NUMA sur lesquelles les performances peuvent être décevantes [89]. Cette faiblesse devrait cependant être corrigée dans la prochaine version du standard.

Cilk

Le projet Cilk [130], développé depuis 1994 au laboratoire du MIT, repose sur un environnement d'exécution et une extension du langage C paramétrée par l'ajout de mots clés spécifiques. L'idée soutenue est de laisser le programmeur se concentrer sur la structure du programme pour en exposer le parallélisme et les affinités, tandis que l'environnement d'exécution prend la responsabilité de l'ordonnancement sur une plateforme donnée. Ce dernier administre les détails d'équilibrage de charge et le choix des protocoles de communication employés.
Relativement simple à mettre en œuvre, Cilk propose un parallélisme à grain très fin et bénéficie d'une gestion extrêmement légère des tâches exprimées par le programmeur. Celles-ci sont organisées sous forme de *pools de tâches* attribués aux processus légers créés au démarrage de l'application. Chaque thread exécute les tâches piochées dans une file qui lui est propre, réduisant ainsi le nombre de synchronisations nécessaires. L'équilibrage de charge est maintenu par un mécanisme de vol de travail. Celui-ci est effectué de façon aléatoire et peut constituer une barrière notable sur les architectures hiérarchiques puisqu'aucune notion de localité n'est prise en compte.

TBB
La bibliothèque TBB [129] (*Thread Building Blocks*) d'INTEL fournit une abstraction de haut niveau pour décrire le parallélisme d'un code C++ destiné à être exécuté sur des architectures multicœurs. Du point de vue du programmeur, l'approche consiste à exprimer les portions de code pouvant être parallélisées sous la forme de tâches TBB récursives. La bibliothèque offre un environnement d'exécution capable de récupérer des informations sur la topologie de l'architecture sous-jacente. Il crée ainsi un nombre de threads égal au nombre de processeurs virtuels de la machine et s'adapte de façon transparente aux différentes architectures multicœurs. La gestion des tâches est très efficace : chaque thread possède sa propre file de travail et l'environnement d'exécution répartit les données à traiter entre les threads de façon transparente pour le programmeur. L'équilibrage de charge se fait par vol de travail pour lequel les informations topologiques sont utilisées pour optimiser la localité des données (utilisation des caches, etc.). La bibliothèque propose également des optimisations réduisant au maximum le nombre de synchronisations et fournit des verrous et opérations atomiques efficaces pour les cas où elles ne peuvent être évitées.

2.1.2 Programmation pour les systèmes à mémoire distribuée

La mémoire distribuée est caractéristique des architectures de grappes, composées de machines interconnectées qui n'ont pas de mémoire commune. Dans ce modèle, chaque processus accède seul à sa mémoire privée. Contrairement au modèle à mémoire partagée, le transfert d'information entre les processus est donc fait par communication explicite.

Pour répondre aux besoins spécifiques des applications, le calcul parallèle sur grappe a été marqué par la création d'interfaces de programmation adaptées à l'utilisation de la mémoire distribuée. En effet, parmi les multiples interfaces et technologies réseau proposées, aucune ne s'est réellement imposée. Il en résulte une grande variété d'interfaces, bas niveau et spécifiques à chaque carte. Leur utilisation depuis l'application engendre la dépendance de celle-ci avec la technologie associée et complexifie sa maintenance, nécessaire à chaque renouvellement réseau ou évolution de l'interface. L'emploi d'une interface dédiée par l'application est de ce fait marginale car non portable, et les programmeurs d'applications adoptent généralement des solutions moins performantes, mais portables et pourvues de garanties de fonctionnalités. Pour répondre à cette demande, des standards de communication par *passage de messages* et des interfaces d'*accès direct à la mémoire distante* ont ainsi été développés.

2.1.2.1 Passage de messages

Le modèle de communication par passage de messages est caractérisé par des échanges explicites de messages entre un émetteur et un récepteur. Ceux-ci font appel à des primitives d'envoi et de réception qui se déclinent selon des modes synchrone ou asynchrone et bloquant ou non bloquant, déterminant les schémas de communication et synchronisation employés. Dans les années 80, les constructeurs ont commencé à vendre des machines parallèles dont l'environnement de programmation était généralement constitué d'un langage de programmation séquentiel (C ou FORTRAN) et d'une bibliothèque de communication par passage de messages pour les communications interprocessus. Pour répondre aux contraintes de portabilité des programmes jusqu'alors dépendants

des bibliothèques spécialisées fournies par les constructeurs, des travaux académiques ont abouti au développement d'interfaces standards pour le passage de messages.

PVM

Une première proposition a vu le jour avec PVM [201] (*Parallel Virtual Machine*) crée en 1989 à l'Oak Ridge National Laboratory et ouvert au public en 1993. La bibliothèque PVM ordonne la création d'une machine virtuelle au-dessus d'un cluster de machines éventuellement hétérogènes et fournit ainsi une abstraction de la structure à mémoire distribuée en une unique machine logique, masquant l'hétérogénéité des technologies sous-jacentes. La machine virtuelle est matérialisée au moyen de démons présents sur chaque nœud et responsables des communications entre ces derniers. Du point de vue du programmeur, tous les processeurs apparaissent comme appartenant à une unique machine. L'interface proposée permet la création de tâches PVM, leur synchronisation et les échanges de messages entre elles. La flexibilité et l'interopérabilité de PVM ont dans un premier temps séduit les programmeurs d'applications parallèles. Cependant, les besoins de performances exigés par le calcul haute performance ont finalement incité les experts à lui préférer le standard MPI, plus élaboré.

MPI

Le standard *Message Passing Interface* [111], annoncé en 1994 par le MPI Forum [110, 113], est l'aboutissement des travaux d'un consortium de constructeurs, d'industriels et d'universitaires, réunis pour proposer une solution standard dédiée au calcul parallèle sur grappes et architectures parallèles. Cette interface de passage de messages définit un ensemble de fonctionnalités et de spécifications génériques, entièrement indépendantes des architectures. Elle s'articule autour de primitives de communications *point-à-point* et d'opérations *collectives*.

La spécification MPI définit un standard pour lequel de nombreuses implémentations sont disponibles. Certaines sont optimisées pour des technologies réseau particulières (MPICH2-MX pour Myrinet, QUADRICS MPI pour QSNET II et MVAPICH [78] pour INFINIBAND), ou pour des architectures spécifiques (BlueGene MPI [60] par exemple). Il existe également des implémentations génériques, dont MPICH2 [62] et OPEN MPI [68, 69], qui présentent des performances légèrement inférieures à celles des implémentations dédiées mais supportent un très large éventail d'architectures et de technologies réseau. Bien que relativement complexe, MPI s'est imposé comme une référence de programmation pour les applications parallèles et a été régulièrement étendu. Une seconde version, MPI-2 [112], a été adoptée en 1997 et la version MPI-3 est prévue pour fin 2011.

2.1.2.2 Accès directs à la mémoire distante

Une alternative à la programmation des architectures à mémoire distribuée par passage de messages consiste à effectuer des accès directs à la mémoire distante, RDMA (*Remote Direct Memory Access*). Ce modèle fait intervenir une phase d'initialisation durant laquelle chaque nœud réserve une zone de son espace mémoire et transmet aux hôtes participants les identifiants relatifs à l'espace réservé. Les échanges de données se font alors par lecture/écriture à distance selon un mode de communication unilatéral (*one-sided*), puisque seul l'initiateur de la communication intervient dans le transfert.

Interfaces et bibliothèques de RDMA
Dans le but de standardiser les communications au-dessus des grappes de calcul, les industriels Microsoft, INTEL et Compaq ont proposé en 1997 l'interface utilisateur VIA [203, 204] (*Virtual Interface Architecture*). Ce standard supporte à la fois le modèle par passage de messages et le RDMA. De plus, il définit un ensemble de fonctions et de structures de données pour un accès direct aux interfaces réseaux (OS-bypass et zero-copie). Cette interface ne s'est jamais véritablement imposée, freinée par une limitation de passage à l'échelle [202] et la concurrence de MPI. La disparition progressive de VIA a conduit à la création en 2002 de l'interface DAPL (*Direct Access Programming Library*). Cette bibliothèque propose des spécifications de niveau noyau (kDAPL) et utilisateur (uDAPL) permettant d'exploiter les capacités RDMA des technologies réseau. Un peu plus populaire, la bibliothèque de communication ARMCI [206] (*Aggregate Remote Memory Copy Interface*) fournit des opérations génériques, efficaces et largement portables de DMA optimisées pour les transferts de données non contiguës en mémoire. Elle se démarque de VIA qui supporte la non contiguïté des données uniquement en mémoire locale, en permettant celle-ci sur la mémoire distante. Elle se soustrait à l'utilisation d'un tampon contigu intermédiaire nécessitant une recopie vers les emplacements mémoire terminaux.

Le modèle de programmation par accès direct à la mémoire distante nécessite l'emploi de mécanismes de détection des données pour contrebalancer l'absence de notification sur le nœud cible. D'un point de vue général, son utilisation est ainsi loin d'égaler la relative simplicité du passage de messages au travers de MPI, dont le modèle se révèle souvent plus adapté à la conception des applications parallèles [205], et qui a largement assis sa prédominance pour l'exploitation des grappes de calcul. Les propriétés matérielles de RDMA offertes par le réseau INFINIBAND, leader actuel du marché des réseaux rapides, constitueraient un support efficace pour l'implémentation de ce modèle. La prédominance écrasante de MPI empiète cependant sur cet axe de développement. Ainsi, plutôt que d'exploiter les propriétés de RDMA matériel directement dans le modèle d'accès direct à la mémoire, les chercheurs les intègrent dans les implémentations MPI.

2.1.3 Mémoire virtuellement partagée

Une seconde alternative à l'utilisation de passage de messages consiste à fournir une abstraction d'un espace d'adressage unique partagé au-dessus de machines interconnectées sans mémoire commune, donnant l'illusion d'une mémoire partagée. On parle alors de mémoire virtuellement partagée ou de DSM (*Distributed Shared Memory*). Les systèmes de DSM reposent sur un support logiciel qui permet à chaque nœud du cluster d'avoir accès en plus de sa mémoire privée, à de la "mémoire partagée". Concrètement, la bibliothèque en charge de ce support maintient une copie de la mémoire partagée dans la mémoire locale à chaque nœud. Elle trace les activités des processus de chaque machine et assure la cohérence et la consistance mémoire en mettant à jour les diverses copies lorsque nécessaire. Ce système adapte ainsi le protocole de cohérence des pages et l'environnement d'exécution intercepte les demandes d'accès distants et génère les requêtes aux hôtes concernés. Les approches basées sur les architectures à mémoire virtuellement partagée ont l'avantage de libérer les programmeurs de la gestion explicite des échanges de données et offrent la possibilité d'utiliser un modèle de programmation à mémoire partagée [123, 124]. Les mécanismes de gestion de la cohérence mémoire sont toutefois coûteux dépréciant souvent l'utilisation des DSM. L'optimisation des programmes sur ces plateformes doit passer par une prise en

compte de la localité des données par rapport aux traitements qui incide sur les défauts de pages, de caches et le faux partage, et s'avère critique sur les performances.

Programmation sur DSM
Plusieurs extensions de langage permettent d'exploiter ce modèle. C'est notamment le cas d'Unified Parallel C [207] (UPC) né de la synthèse d'extensions antérieures (AC, Split-C et PCP). UPC présente un espace d'adressage unique et partitionné, assisté d'un modèle de cohérence mémoire dans lequel les variables peuvent être lues et écrites par différents threads, mais dont la localisation physique est associée à un unique processeur. Le programmeur a la charge d'expliciter les variables à partager, et le langage s'appuie sur le modèle de mémoire virtuellement partagée pour distribuer les calculs en fonction de l'emplacement physique des données. Pour alléger les coûts de gestion de la cohérence mémoire, UPC propose en addition à la *cohérence mémoire séquentielle*, qui entraîne une synchronisation de la mémoire pour chaque écriture/lecture, une *cohérence relâchée*, pour laquelle la mémoire n'est harmonisée que lors des synchronisations explicites du programme (verrous, barrières...).

Pour étendre l'utilisation du standard OPENMP, initialement restreint à l'exploitation de machines à mémoire partagée, INTEL a mis au point Cluster OPENMP [208], qui s'appuie en interne sur la DSM Tread Marks. Ce standard diffère d'OPENMP par le mot-clé *sharable* qui déclare des variables utilisées par plusieurs threads et dont la cohérence doit être assurée pour le bon fonctionnement du programme. Définies explicitement par le programmeur ou implicitement par le compilateur, ces variables sont groupées au sein de pages protégées sur lesquelles un accès génère l'envoi d'un signal. Ce dernier est intercepté par la bibliothèque qui produit alors la requête appropriée pour la mise à jour de la page. L'emprise de ce mécanisme coûteux peut être réduite par le modèle relâché d'OPENMP et par une gestion favorable à la localité pour limiter les défauts de pages depuis les nœuds distants, mais doit être conjecturée par le programmeur.

2.1.4 Modèles de programmation hybrides

Les grappes de calcul, composées de nœuds multicœurs et multiprocesseurs interconnectés par une infrastructure réseau, possèdent un modèle mémoire hybride. L'utilisation de l'interface MPI domine la programmation des applications destinées aux clusters, cependant des travaux remettent en cause l'emploi de ce modèle qui n'est pas forcément le plus adapté aux nouvelles architectures. En effet, la multiplication du nombre de processeurs et de cœurs par machine s'accompagne d'une augmentation du nombre de processus MPI, et donc d'une réduction de la taille du problème traité par chacun. Lorsque cette taille devient trop faible, le coût de gestion des processus et l'augmentation des accès concurrents aux ressources entravent les performances. De plus, l'utilisation du modèle par passage de messages sur les nœuds de calcul disposant de mémoire partagée apparaît peu judicieux.

Pour résoudre ce problème de passage à l'échelle, des chercheurs explorent des pistes d'utilisation combinée (*hybride*) de plusieurs modèles générant moins de processus MPI par nœud. Ils proposent ainsi l'utilisation de threads à l'intérieur du code MPI pour exploiter localement le parallélisme, par exemple basé sur la bibliothèque de threads POSIX ou sur OPENMP [127, 119].

Ces modèles hybrides peuvent permettre une meilleure exploitation des modèles mémoire que l'on peut trouver dans les grappes de calcul et sont ainsi de plus en plus populaires. Ils requièrent toutefois la gestion conjointe de modèles distincts et donc un investissement important de la part du programmeur. Leur utilisation reste ainsi encore minoritaire comparée à celle de MPI seul. Les bénéfices des modèles hybrides sont variables et dépendent de multiples facteurs tels que le degré de parallélisme et la granularité choisie, la plateforme et les coûts de communication ou encore les schémas d'accès mémoire [120, 121, 122].

2.1.5 Bilan

Différents modèles de programmation peuvent être mis en œuvre pour exploiter les architectures parallèles en fonction de leur modèle mémoire (mémoire partagée et distribuée). Ils sont utilisés au travers d'interfaces et de standards dédiés permettant d'assurer un certain confort de programmation et la portabilité des applications.

De nombreuses évaluations comparatives discutent des performances des différents modèles de programmation proposés [144, 200, 126, 125]. Elles démontrent le caractère variable des performances de ces modèles selon le contexte applicatif et les architectures cibles. En pratique, aucun modèle ou standard n'est meilleur que les autres d'un point de vue général. Le choix d'un "bon" modèle de programmation est ainsi spécifique à chaque application et réside dans un compromis entre l'affinité avec le schéma applicatif, les architectures cibles, les critères d'efficacité requis et l'investissement nécessaire de la part des programmeurs.

Les grappes de calcul qui dominent le paysage du calcul haute performance présentent un modèle à mémoire distribuée mais dont chaque nœud dispose de mémoire partagée. Bien que celui-ci ne soit pas forcément le plus adapté, la popularité du standard MPI le maintient au stade de favori pour l'exploitation des grappes de machines multicœurs. Il est ainsi souvent utilisé de façon homogène pour les transferts inter-nœuds et intra-nœud. Créé depuis plus de 15 ans, MPI est bien implanté dans la communauté scientifique et fait l'objet de nombreuses implémentations. Celles-ci sont le résultat de recherches importantes et s'appuient sur de multiples techniques de transferts et algorithmes de communication développés au fil des ans.

2.2 Gestion des communications dans les bibliothèques MPI

Le standard MPI [110, 113] est devenu la référence pour les communications inter-nœuds sur grappe et a été largement plébiscité. L'absence de modèle de programmation réellement adapté aux architectures multicœurs majore son rôle au sein des machines de calcul. MPI est ainsi fréquemment utilisé sur les clusters de machines scientifiques tant pour les échanges inter-nœuds qu'intra-nœud. En réponse à cette utilisation et au déploiement des machines employées dans les grappes de calcul, les implémentations MPI ont évolué pour proposer des transferts et des algorithmes de communication toujours plus performants.

2.2.1 Stratégies de transferts utilisées par les bibliothèques MPI

Des transferts réseau performants

Pour assurer l'efficacité des transferts réseau, les implémentations MPI s'appuient sur des pilotes matériels performants et profitent des techniques spécifiques aux cartes réseau haute performance évoquées en Section 1.1.2. La Figure 2.1 présente les débits obtenus sur un réseau MYRI-10G pour l'exécution du test IMB Pingpong [31] au-dessus de la bibliothèque OPEN MPI et en comparaison avec le *microbenchmark* constructeur `mx_pingpong`. L'implémentation générique OPEN MPI est conçue pour supporter de nombreuses architectures et interfaces réseaux. L'utilisation de MYRI-10G peut se faire au travers de deux briques logicielles OPEN MPI [1]. Le module MTL commande une utilisation quasi directe du pilote MX et offre des performances comparables à celles du test `mx_pingpong` utilisé comme référence (moins de 170 ns de surcoût de latence, 5% de perte maximum de débit). Le module BTL fournit quant à lui une interface uniforme pour l'utilisation des différents pilotes et impose des compromis à l'origine d'un coût supplémentaire de la pile logicielle (latence de 280 ns supérieure à celle du test constructeur). OPEN MPI offre ainsi la possibilité de choisir entre un module réseau dédié offrant des performances quasi optimale et un module générique dont les performances ne sont que légèrement inférieures.

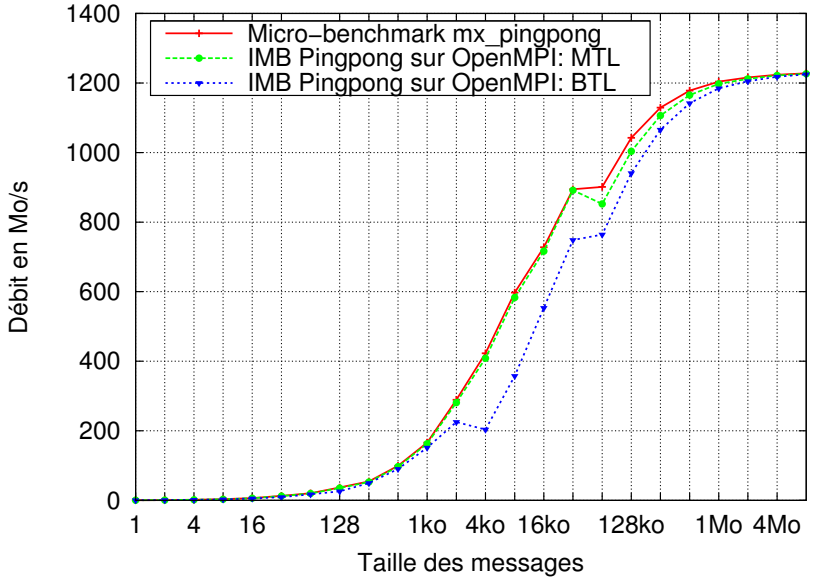


FIGURE 2.1 – *Débit d'un ping-pong en fonction de la taille des messages sur réseau Myri-10G pour le microbenchmark* `mx_pingpong` *et le test IMB Pingpong au-dessus de* OPEN MPI.
Le module MTL permet une utilisation quasi directe du pilote MX.
Le module BTL fournit une interface uniforme pour de nombreux pilotes.

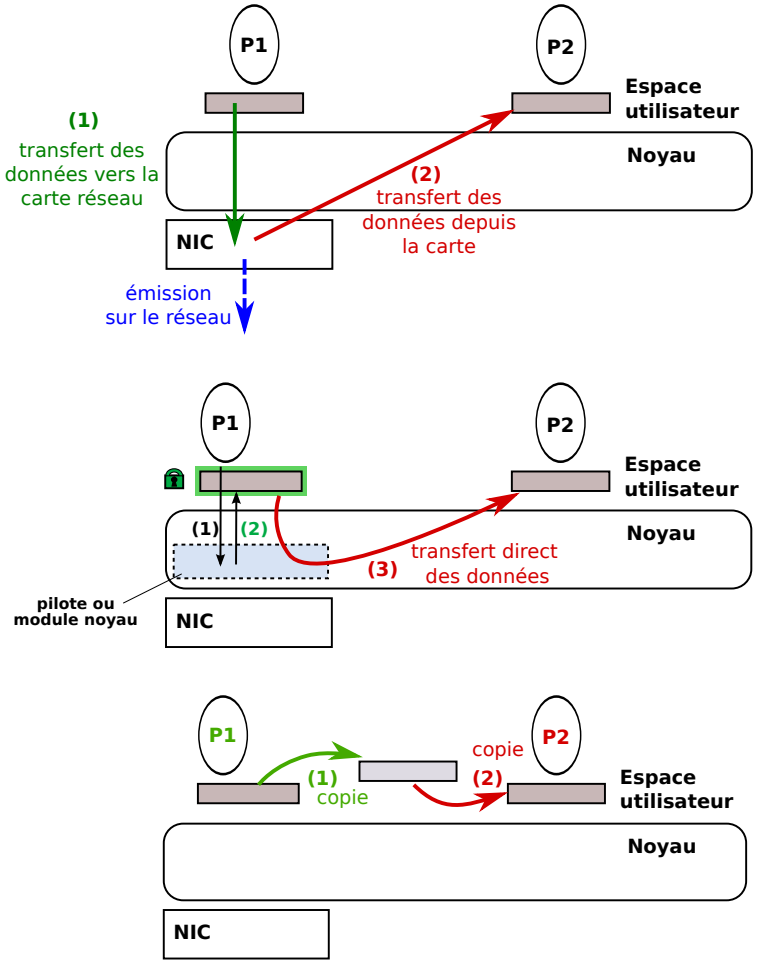
D'un point de vue général, l'utilisation efficace des réseaux rapides dans le cadre des communications MPI a été largement explorée, permettant aujourd'hui aux bibliothèques MPI d'assurer les transferts réseau avec de très faibles surcoûts logiciels. Elles peuvent également profiter des

1. Une représentation partielle de la pile OPEN MPI est illustrée en Figure 5.3.

avantages matériels des interfaces supportées. La bibliothèque spécifique MVAPICH, reconnue comme pilier de la recherche expérimentale sur INFINIBAND, est par exemple capable d'exploiter le RDMA propre à cette technologie [80].

Stratégies de transfert en mémoire partagée

La présence de machines multiprocesseurs et multicœurs au sein des clusters entraîne l'utilisation de plusieurs processus MPI par nœuds de calcul. Les processus locaux à une machine communiquent par le biais de transferts intra-nœud (dits en *mémoire partagée* en référence au modèle mémoire de la machine et par opposition aux transferts réseau). Ces transferts s'appuient sur des mécanismes spécifiques mis en œuvre par les implémentations du standard MPI qui sont généralement intégrés dans leur pile logicielle sous forme de modules [68, 69] ou de sous-systèmes spécifiques [63, 64, 65]. Les implémentations MPI mettent ainsi en œuvre de multiples techniques de transferts.



(a) Loopback matériel.

(b) Transfert direct avec assistance du noyau.

(c) Utilisation d'un tampon en mémoire partagée.

FIGURE 2.5 – *Techniques de communication intra-nœud.*

Historiquement, les communications inter-nœuds et intra-nœud utilisaient le même mécanisme de transfert. Comme illustré en Figure 2.5 (a), les messages étaient transmis à la carte réseau (1), qui prenait en charge leur acheminement par émission sur le réseau ou par réinjection des données (2) grâce à une boucle locale (*loopback*). Ce modèle de communication homogène et portable est rapidement apparu insuffisant dans le cas de communications locales pour lesquelles il génère un surcoût de transfert des données sur le bus d'entrées-sorties et une surcharge de celui-ci [2].

Des chercheurs ont exploré la possibilité de détourner les propriétés de copies directes (*Direct Memory Access*) des interfaces réseau, au sein de protocoles dédiés grâce à l'assistance du noyau (Figure 2.5 (b)). Alors que la bibliothèque MPI invoque la primitive de communication sans avoir conscience de la localité du destinataire (1), le protocole détermine s'il s'agit d'un transfert externe ou interne. En communication intra-nœud, il punaise ainsi la mémoire nécessaire avec l'assistance des pilotes de la carte réseau (2) profitant des accès privilégiés de ceux-ci pour effectuer un transfert direct entre processus locaux (3). Ces piles logicielles (extension de pilote), telles que GM [53], BIP [57, 58] ou MX [54] sont spécifiques au matériel réseau employé et ne sont donc pas portables. Plus récemment, cette idée a ainsi été extraite et implémentée de façon générique sous la forme de modules noyau, tels que LIMIC2 [79, 82], invoqué dans MVAPICH, et KNEM [3], généralisé depuis OPEN-MX [115, 116] et employé dans les implémentations MPICH2 [66] et OPEN MPI [116].

Une solution plus portable utilisée dans les implémentations MPI profite de la présence de mémoire partagée comme intermédiaire de communication entre les processus [69, 63, 81] (Figure 2.5 (c)). Un processus émetteur copie ses données dans une zone de mémoire partagée (1) depuis laquelle le processus récepteur les recopie vers l'emplacement mémoire destination (2). Cette technique a l'avantage de bénéficier de l'utilisation du cache pour des messages de petite et moyenne taille pour lesquels ils offrent de bonnes performances [142], mais nécessite l'intervention des deux processus communicants et impose deux copies mémoires.

Le développement des systèmes d'exploitation a également contribué à la définition de nouvelles méthodes de transfert intra-nœud. L'appel système `vmsplice` [16], introduit dans le noyau Linux 2.6.17, permet par exemple au processus récepteur de copier directement les données attachées à un tube UNIX par l'émetteur. Le support SMARTMAP [145] profite quant à lui de la gestion mémoire "légère" du système d'exploitation *catamount* des machines Cray. Il propose ainsi des copies directes entre processus à très faible surcoût grâce à une projection facile de la mémoire.

Comme nous le verrons au Chapitre 6, ces méthodes offrent des performances plus ou moins bonnes en fonction des architectures et des caractéristiques du transfert [142] et peuvent ainsi être utilisées de façon combinée. L'utilisation de ces différentes techniques en accord avec la structure matérielle est ainsi la piste de recherche principale pour l'amélioration des transferts en mémoire partagée.

2. Une analyse [50] suggère que les dernières générations de cartes réseau connectées au travers de bus PCIe récents présentent de telles performances que l'utilisation de la boucle locale réseau est à nouveau compétitive. Cette technique surcharge cependant les bus d'entrées-sorties alors que seul le bus mémoire devrait être utilisé, aspect problématique avec la multiplication des cœurs.

3. `http://runtime.bordeaux.inria.fr/knem/`

2.2.2 Opérations collectives et réduction du trafic interprocessus

Contrairement aux communications point-à-point établies entre paires de processus, les opérations collectives engagent un ensemble plus large de tâches. Elles sont ainsi caractérisées par des échanges multiples effectués entre plusieurs paires de processus et fonctions du schéma de communication invoqué (Figure 2.6) et des algorithmes qui le mettent en œuvre. Les différents schémas de communication comportent notamment des opérations avec :

a) un émetteur pour de multiples récepteurs (*one-to-all*),
b) de multiples émetteurs pour un récepteur (*all-to-one*),
c) de multiples émetteurs et récepteurs (*all-to-all*).

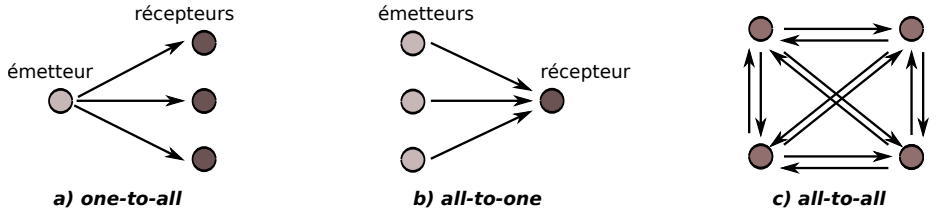


FIGURE 2.6 – *Schémas de communications collectives.*

Les opérations collectives profitent évidemment des perfectionnements apportés aux transferts point-à-point. La durée de ces communications complexes et intensives est toutefois principalement dictée par le nombre, le coût, et l'interdépendance logique des échanges, ainsi que par les contentions générées.

Pendant longtemps, améliorer les performances des collectives MPI consistait à réduire le nombre de transferts inter-nœuds coûteux au profit des communications intra-nœud. Des approches considèrent également les caractéristiques de la structure réseau (bandes passantes, latences et topologie) pour éviter les transferts les plus coûteux notamment sur des systèmes à grande échelle [155, 153, 151, 152, 154]. La plateforme MagPIe [151] adapte par exemple le schéma des opérations sur un environnement de grappes hautement hiérarchiques en minimisant les quantités de données échangées sur les liens les plus lents.

L'introduction des machines multiprocesseurs au sein des grappes exige des échanges intra-nœud, inter-nœuds, ou des deux catégories, selon la répartition des tâches participant à l'opération collective. La mise au point du modèle par mémoire partagée a donné lieu à la création d'algorithmes multi-protocoles. Sistare *et al.* ont introduit dans leurs algorithmes de communication l'utilisation de mémoire partagée [188] précédemment étudiée dans le cadre de transferts point-à-point. Ils évitent ainsi le *passage de messages intra-nœud* et réservent ce modèle aux échanges réseau.

La littérature scientifique regorge d'un vaste éventail d'algorithmes de collectives développés sur la théorie des graphes pour réduire le trafic interprocessus, et dont les performances dépendent de l'adéquation du schéma des transferts avec la structure de l'architecture cible. Les chercheurs ont ainsi constitué de véritables catalogues d'algorithmes [4] qui proposent des schémas plus ou moins bien adaptés aux architectures et aux caractéristiques des transferts (tailles des messages,

4. Certains d'entre eux sont décrits dans [158, 162, 161, 159, 168, 169, 167, 170, 173, 160, 188, 171, 166, 172, 163, 164, 165].

utilisation de mémoire partagée, etc.). Les implémentations MPI s'appuient sur ces différents algorithmes utilisés de façon complémentaire pour minimiser le trafic nécessaire en fonction des propriétés des communications invoquées.

2.2.3 Affiner les algorithmes de communication

L'élaboration d'implémentations MPI efficaces a favorisé le développement de nouveaux algorithmes orientés vers l'exploitation du potentiel des nouvelles interfaces réseau et d'une utilisation plus fine de la mémoire partagée.

Prendre en compte les nouvelles propriétés réseau
L'évolution des réseaux dans les grappes de calcul a suscité un regain d'intérêt pour la recherche d'algorithmes attentifs aux propriétés réseau. Kumar *et al.* [182] proposent par exemple des opérations *all-to-all* adaptées aux architectures multicœurs qui basent leurs communications inter-nœuds sur les performances des différentes interfaces INFINIBAND. Kandalla *et al.* [181] proposent des algorithmes hiérarchiques pour les opérations de rassemblement (*gather*) et de distribution (*scatter*) qui tiennent compte de la topologie réseau des clusters possédant plusieurs niveaux de switch. Coti *et al.* [196] ont conçu des algorithmes hiérarchiques pour grille exploitant des opportunités d'optimisation en mémoire partagée, et Matsuda *et al.* [156] revisitent des algorithmes de collectives hiérarchiques pour les adapter aux technologies réseau efficaces qui interconnectent les clusters dans les grilles modernes et éviter les congestions sur ces liens.

Plusieurs contributions sur l'implémentation MVAPICH tirent parti des caractéristiques matérielles d'INFINIBAND pour améliorer les performances obtenues sur les grappes de calcul. En plus de profiter des transferts RDMA [194, 193] elles exploitent par exemple le multicast matériel pour enrichir les implémentations des opérations collectives [193, 190, 191, 192, 189].

Une exploitation plus efficace de la mémoire partagée
Avec la multiplication des unités de calcul au sein des nœuds, la quantité de transferts intra-nœud augmente et leur impact sur les performances globales en est accru [36]. Des efforts ont été fait pour proposer des algorithmes et stratégies optimisés pour exploiter efficacement la mémoire partagée. Ceux-ci sont le plus souvent intégrés au sein de communications hiérarchiques multi-canaux. Le lien entre le *canal réseau* employé pour les échanges inter-nœuds et le *canal mémoire partagée* des échanges intra-nœud, est alors effectué par l'intermédiaire d'un processus local à chaque nœud appelé *racine locale* (*local root*), *processus maître* ou encore *leader*. Dans le cas d'un schéma one-to-all, illustré Figure 2.7, les données sont transmises depuis le processus racine de l'opération aux processus locaux en mémoire partagée (1), et au travers du réseau vers les nœuds participants (2). Sur chacun d'entre eux, le *leader* récupère les données puis les diffuse aux autres processus du nœud (3). Les communications intervenant sur le canal réseau peuvent ainsi être soumises à des algorithmes hiérarchiques tenant compte de la structure et des caractéristiques des réseaux, et les algorithmes spécialisés pour la mémoire partagée prennent ensuite le relais.

Wu *et al.* [185, 187] généralisent l'approche de Sistare [188] en proposant un ensemble de primitives génériques qui classifient les accès à la mémoire partagée et sont utilisées comme briques de

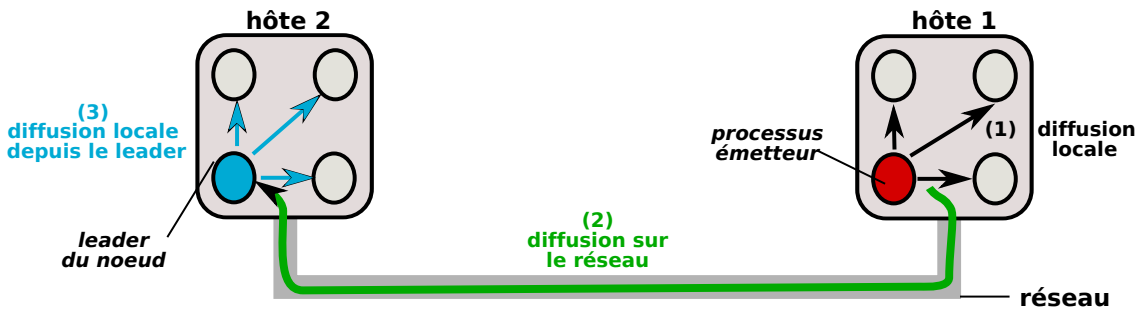


FIGURE 2.7 – *Communication one-to-all sur deux nœuds disposant de 4 processus chacun.*

construction pour les opérations collectives. Selon leur proposition, un *broadcast* est ainsi construit à l'aide de trois primitives : *Sender_put()* qui permet à l'émetteur d'écrire dans le tampon partagé ; *Group_sync()* en charge de la synchronisation d'un groupe de processus ; et *Group_get()* permettant aux processus récepteurs de récupérer un message depuis le tampon. Ils proposent un modèle de programmation pour des grappes de machines SMPs [186] profitant des accès concurrents à la mémoire et de mécanismes de recouvrement des communications intra et inter-nœuds.

L'équipe du professeur Panda, de l'Université de l'état de l'Ohio, a développé de nombreuses optimisations pour l'implémentation MVAPICH. Les contributeurs ont notamment exploré le potentiel d'utilisation combinée de la mémoire partagée locale et distante [184, 183, 194]. Les caractéristiques du réseau INFINIBAND leur permettent d'effectuer des transferts inter-nœuds efficaces par accès mémoire distants (*Remote Direct Memory Access*). Ils appuient entièrement leurs transferts sur le modèle de mémoire partagée et bénéficient du partage des buffers de données, accessibles pour les communications internes au nœud comme pour les communications externes. Ils s'affranchissent ainsi des copies multiples induites par l'utilisation traditionnelle de canaux distincts pour les communications intra et inter-nœuds, et profitent d'un meilleur recouvrement entre ces deux types de communications.

2.2.4 Bilan

La popularité de MPI a favorisé le développement d'implémentation performantes. La collaboration des tâches MPI nécessite de nombreux échanges assurés par transferts point-à-point ou opérations collectives. La recherche florissante dans ce domaine a conduit au développement de très nombreux algorithmes de communication minimisant le trafic interprocessus. Les chercheurs ont travaillé à la réduction du coût de traversée de la pile logicielle pour exploiter au mieux les capacités des réseaux rapides, assurant des transferts réseau aujourd'hui quasi optimaux et développé diverses méthodes pour les échanges internes aux machines. L'optimisation des transferts intra-nœud est aujourd'hui axée sur une combinaison intelligente des stratégies disponibles qui exhibent des performances différentes en fonction du contexte de communication, mais aussi de spécificités matérielles des architectures contemporaines. En effet, l'aspect hiérarchique des machines modernes présente des caractéristiques variables impactant les performances des échanges entre les tâches collaborantes au sein d'une machine. C'est ainsi à une gestion fine de ces spécificités que les chercheurs doivent maintenant s'attacher pour accroître l'efficacité des communications.

2.3 Impact des architectures contemporaines

Comme nous l'avons vu en Section 1.2 les architectures des machines contemporaines sont de plus en plus complexes et hiérarchiques. L'organisation interne (*topologie*) des machines est connue pour avoir des effets nocifs sur les performances des applications parallèles dès lors qu'elle n'est pas finement prise en compte. Cette section retrace un ensemble de ces effets caractéristiques nécessitant une considération et un traitement soigné pour assurer un bon niveau d'efficacité sur ces architectures. Elle expose également les moyens à disposition des programmeurs pour détecter et tenir compte de la topologie des calculateurs modernes ainsi que les pistes de recherche récentes permettant de répondre automatiquement aux contraintes qu'elles génèrent.

2.3.1 Quels défis pour les programmeurs ?

L'accès uniforme à une mémoire monolithique dans les machines SMP permettait l'utilisation d'un modèle de programmation simple et plat à l'origine de leur succès. Sur les machines multicœurs et NUMA, les performances sont obtenues grâce à la distribution des bancs mémoire, qui limite la contention sur le bus mémoire, et à l'utilisation des caches qui réduisent les temps d'accès mémoire et permettent un partage efficace de données entre des tâches exécutées sur des cœurs dotés d'un cache commun. Ces architectures sont ainsi marquées par de fortes contraintes de localité entre les tâches et les données, amplifiées par les niveaux multiples de hiérarchie.

Effets de cache

Une utilisation pertinente de la mémoire cache disponible sur les processeurs est essentielle à la rapidité des exécutions. Cette propriété n'est pas nouvelle et depuis longtemps prise en compte par les programmeurs, conscients des risques de la pollution de cache et du faux partage [11], ainsi que des bénéfices de la réutilisation des données contenues dans le cache [199, 19].

Dans le cas des architectures multicœurs disposant de caches partagés, la distribution des tâches sur des cœurs partageant un cache peut favoriser les performances lorsque les tâches travaillent sur les mêmes données, ou dans le cas contraire générer une concurrence sur le cache pénalisante. Les travaux de Chai *et al.* [36] mettent ainsi en évidence l'efficacité des communications inter-processus obtenue pour les petits et moyens messages grâce au partage de cache, mais aussi le goulot d'étranglement que peuvent représenter ces mêmes caches et les accès mémoire lorsque la réutilisation des données en cache n'est pas mise en œuvre. L'étude de Fedorova *et al.* [20] analyse les défauts de cache conséquents à une trop forte concurrence de threads sur un même cache partagé. Elle démontre ainsi l'intérêt de répartir les threads sur d'autres puces ou de revoir l'ordonnancement des threads pour rendre l'exécution plus équitable. Les travaux de Song [27, 26] proposent un modèle analytique pour prédire le nombre de défauts sur un cache L2 partagé pour les données privées et partagées, à partir d'une trace d'utilisation du cache par un thread.

Effets NUMA et contention

La répartition de la mémoire en plusieurs bancs sur les architectures NUMA exige un placement et un ordonnancement des tâches soignés. En effet, un thread doit être placé près des données

qu'il manipule pour éviter les accès mémoire distants, dont le surcoût peut être pénalisant pour les performances. Dans leur travaux, Timothy Brecht [34] et Antony *et al.* [42] démontrent ainsi l'importance des décisions de placement des tâches sur les architectures NUMA, dont le facteur varie selon les architectures et augmente avec la taille de la machine.

Pour illustrer le potentiel d'impact NUMA, la Table 2.1 présente les bandes passantes de copies mémoire de données placées sur les différents nœuds de la machine *Dancharia* (représentée Figure 2.8), et initiées depuis le nœud 0. Alors que l'accès à la mémoire locale offre dans cet exemple une bande passante de 5119 Mo/s, on observe en comparaison une bande passante très inférieure pour les accès distants, avec 39% de dégradation pour un lien d'interconnexion HYPERTRANSPORT traversé, 59% pour une distance de 2 liens et jusqu'à 68% pour 3 liens. Le débit observé pour le nœud 5 se trouve à mi-chemin entre les débits attendus pour une distance de 2 et 3. Cela est dû à une caractéristique du routage HYPERTRANSPORT qui définit des chemins différents pour les requêtes mémoire (chemin 0-2-5) et les réponses (5-3-1-0), donnant une distance factice de 2,5.

Nœud	0	1	2	3	4	5	6	7
Distance NUMA	0	1	1	2	2	2	3	3
Bande passante (Mo/s)	5119	3112	3155	2084	2085	1809	1794	1649

TABLE 2.1 – *Bande passante des copies mémoire du benchmark Stream [30] effectuées depuis le nœud 0, en fonction de l'emplacement des données.*

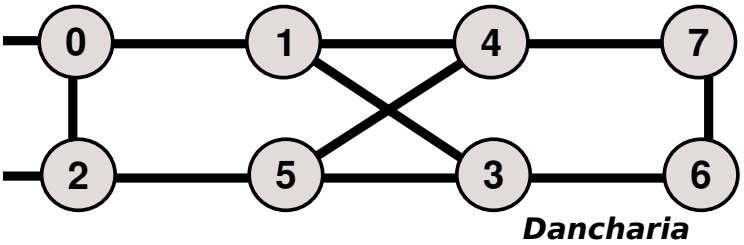


FIGURE 2.8 – *Topologie NUMA de la machine Dancharia composée de 8 nœuds hexa-cœurs Opteron et détaillée en Annexe A.3.*

Le placement est également un facteur déterminant pour les contentions mémoire. La concurrence sur les liens peut être source de dégradations importantes de la latence et de la bande passante des transferts, directement répercutées sur les performances applicatives. La contention est de plus difficilement modélisable ou prédictible. Elle dépend du trafic généré par les applications, du placement des tâches et des données, du routage des requêtes, de l'efficacité de l'usage des caches, du protocole de cohérence, ainsi que de spécificités matérielles plus ou moins détectables. L'étude de Tuduce *et al.* [198] évalue par exemple le partage de bande passante mis en jeu sur des bi quadri-cœurs Xeon E5345. Elle détecte un partage équitable de bande passante sur les bus mémoire et les contrôleurs de bus, mais un partage inégal entre les cœurs. Les ralentissements d'exécution dus aux contentions mémoire ne sont donc pas toujours proportionnels à la demande de chaque tâche.

Combinaison des caractéristiques : la nostalgie des SMPs

De multiples analyses s'attachent à caractériser les architectures modernes. Elles expriment des

effets variables selon les structures matérielles et les applications étudiées, mais sont unanimes sur le rôle critique de la prise en compte des affinités au-dessus de la topologie. Juckeland *et al.* présentent par exemple une évaluation des effets NUMA et des hiérarchies de caches sur serveurs SGI Altix [22] et Yang *et al.* évaluent les conséquences du placement des threads et de la mémoire sur les serveurs SunFire X4600 M2 [18]. Leur analyse illustre les pénalités encourues suite à une mauvaise répartition NUMA des tâches et de la mémoire, mais aussi l'importance considérable que peut avoir l'utilisation du cache sur les performances applicatives. Les structures des architectures contemporaines affectent bien sûr les différents modèles de programmation [119], qu'il s'agisse par exemple de communications MPI intra-nœud [99] ou des communications en mémoire partagée [37, 35].

La Figure 2.9 présente un exemple concret des effets de cache et NUMA sur les performances des transferts de données sur la machine de la plateforme *Bertha* (détaillée en Annexe A.9). Bertha est composée de quatre nœuds NUMA comportant chacun quatre puces Intel Xeon hexa-cœurs Dunnington X7460 dont une représentation est proposée en Figure 2.10. Les courbes correspondent au débit de lecture obtenu avec l'outil LMbench [32] sur des données allouées sur le nœud NUMA local (courbe rouge) ou sur un nœud distant (courbe verte).

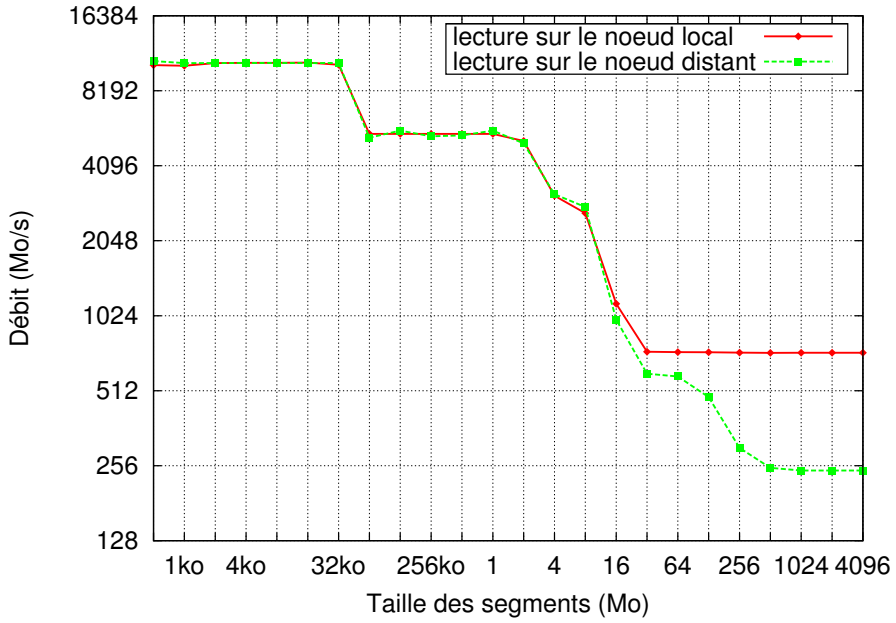


FIGURE 2.9 – *Débit de lecture mémoire en fonction de la taille des données pour une allocation mémoire sur différents nœuds de la machine Bertha.*

Les effets de cache sont particulièrement visibles. Tant que la taille des données est inférieure à la taille d'un cache, la lecture faite depuis celui-ci est particulièrement efficace. Les performances sont alors relatives à la vitesse d'accès au cache contenant les données et ce qui explique les différents échelons de chaque courbe. Le débit optimal est ainsi obtenu pour une taille allant jusqu'à 32 ko alors que les données lues sont contenues dans le cache L1. Au delà, les performances chutent pour atteindre un second palier correspondant à l'utilisation du cache L2 d'une taille de 3 Mo. Après ce seuil, le cache L3 prend le relais et assure les performances des transferts

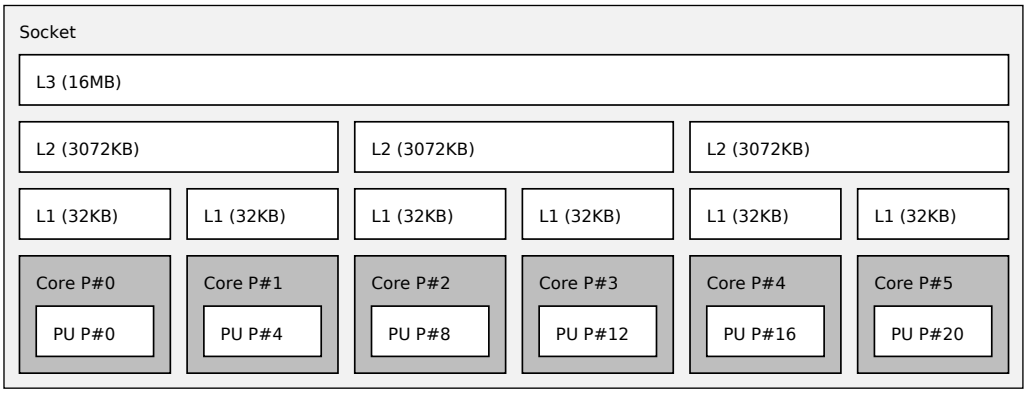


FIGURE 2.10 – *Processeur Intel Xeon hexa-cœur Dunnington X7460 utilisé dans* Bertha

de segments inférieurs à 16 Mo. Au delà de ce stade, les données ne tiennent plus dans les caches du processeur et sont lues directement sur les bancs mémoire. La lecture sur le banc local offre un débit de 725 Mo/s tandis que celle sur un nœud distant est réduite à 245 Mo/s, dénotant un fort effet NUMA sur la bande passante avec un facteur légèrement inférieur à 3. Lorsque les données sont placées sur un nœud distant, la courbe marque un palier supplémentaire pour une taille de données entre 16 et 256 Mo. Celui-ci est dû à une particularité de notre machine de test. En effet, les contrôleurs mémoire IBM de cette machine sont capables d'émuler un niveau de cache supplémentaire. Une zone mémoire de chaque nœud est réservée pour produire un cache "L4" dont la fonction est de conserver une copie des données allouées sur un nœud distant et accédées depuis le nœud local. Ce niveau de cache simulé permet ainsi de s'affranchir des coûts d'accès répétés sur un banc mémoire distant et explique ce palier.

Les notions d'affinité et de localité entre les tâches et les données sont ainsi au cœur d'une exploitation efficace des structures hiérarchiques. Elles complexifient largement les efforts à fournir et amènent les chercheurs à regretter les machines SMPs plates. De nombreuses études sur les effets du placement des tâches (processus, tâches applicatives, flots d'exécution, *threads*) et de la mémoire sur les machines multicœurs et NUMA ont été menées, notamment dans le contexte de l'ordonnancement. Des travaux exposent ainsi l'importance de la localité des données [91, 92, 93, 94], ou encore l'intérêt de faire migrer des pages pour conserver les affinités mémoire [95, 89]. L'évolution matérielle réactualise sans cesse les problématiques d'affinités apportant avec chaque génération de nouvelles contraintes à dénouer.

2.3.2 Quels supports pour détecter et exploiter la topologie des architectures ?

Une première gestion des affinités a été entreprise au niveau de politiques d'ordonnancement grâce à l'emploi de plusieurs listes de threads favorisant un réordonnancement "local" [96, 97]. Certaines sont organisées hiérarchiquement et incluent un niveau NUMA pour guider un ordonnancement préférentiel des tâches sur le nœud contenant à priori leurs données [98]. Ces politiques s'accompagnent de stratégies d'allocation mémoire, locales au premier accès (*first-touch*) ou cycliques, page par page sur les nœuds (*round robin*), qui se comportent plus ou moins bien selon les applications. Des cas pathologiques sont connus pour générer des allocations particulièrement in-

appropriées qui majorent les contentions sur les bancs mémoire [5]. Pour compenser ces stratégies simplistes, certains systèmes proposent des mécanismes de migration mémoire automatique appuyés sur des compteurs de performances matériels. Ils détectent les accès distants répétés aux pages et déplacent celles-ci près des tâches qui initient ces accès. Ces dispositifs requièrent des analyses imposantes dont le coût croît avec la précision des statistiques. Ils induisent ainsi une consommation de ressources pouvant mitiger, voire même inverser leur effet. En l'absence d'information sur le déroulement des applications, la pertinence des déplacements peut de plus être modérée [84, 87]. La migration automatique est ainsi parfois désactivée par les programmeurs.

Ces techniques de répartition de tâches et de données ne sont pas suffisantes pour répondre seules aux contraintes d'affinités exprimées par les multiniveaux de hiérarchie des architectures contemporaines et ne peuvent s'adapter de manière générique aux différents types d'applications. Les programmeurs sont généralement les seuls à disposer des informations sur le déroulement d'un programme permettant de calculer une répartition judicieuse des tâches et des données, cohérente avec leurs affinités. Il n'est ainsi pas rare que ceux-ci prennent eux-mêmes en charge la distribution des tâches et des données au travers de la machine. Une telle démarche assure de bonnes performances sur une machine cible donnée, mais se fait au prix d'efforts d'analyse de la topologie et au détriment de la portabilité. De nouvelles pistes cherchent ainsi à raffiner l'ordonnancement et la gestion mémoire sur les topologies actuelles. De tels travaux ont pour prérequis la détection de l'architecture matérielle et la disponibilité d'outils permettant de guider le placement des tâches et des données.

2.3.2.1 Détecter la topologie et forcer le placement des tâches

Découvrir la topologie d'une machine pour effectuer un placement manuel depuis une application est un problème difficile. Le matériel exporte peu d'informations et les indications fournies par les systèmes d'exploitation sont limitées. Celui-ci peut par exemple expliciter le nombre de nœuds d'une machine sans détailler la distance entre eux. Des bibliothèques propriétaires savent adapter le nombre de tâches à celui des unités de calcul ou exploiter d'autres aspects de la topologie, mais leur utilisation est restreinte à des architectures spécifiques [129, 128]. Pour un utilisateur, appuyer un code sur la topologie d'une machine revenait ainsi le plus souvent à explorer les informations exposées par exemple dans le système de fichiers virtuels `sysfs` de Linux, ou plus contraignant encore, à étudier le manuel de la machine cible. En réponse à ces difficultés, plusieurs outils de placement et détection de topologie ont été mis au point.

CPU affinity

Parmi les outils de placement disponibles, l'interface de programmation *CPU affinity* de la bibliothèque C standard permet d'expliciter le placement des processus sur les unités de calcul de la machine. Le programmeur manipule des masques de processeurs (`cpu_set`) autorisant l'exécution d'une tâche sur le sous-ensemble de processeurs déterminé. Cette technique assure ainsi le place-

5. Par exemple, une stratégie d'allocation *first-touch* effectuée sur toutes les pages par un unique thread au cours d'une phase d'initialisation conduit à des accès distants par l'ensemble des threads exécutés sur des nœuds NUMA différents. Une politique de type *round-robin* peut aboutir à une allocation NUMA des pages en décalage par rapport au placement NUMA des tâches qui accèdent majoritairement à chacune d'entre elles.

ment des tâches mais exige du programmeur de s'informer sur l'organisation des unités de calcul dont la numérotation peut varier selon les architectures et les systèmes.

libnuma
Des systèmes d'exploitation propriétaires tels que IRIX [90] et Solaris [23] disposent depuis longtemps de supports NUMA permettant de tenir compte de la localité NUMA dans les choix d'allocation ou de migration mémoire et de placement des tâches. Plus récemment, le système d'-exploitation LINUX a été doté de l'outil de placement `numactl` qui permet de forcer le placement des tâches et de la mémoire sur certains nœuds. En espace utilisateur, les applications profitent de la bibliothèque associée `libnuma` [9] pour agencer les threads et la mémoire en fonction de leur affinité.

PLPA
Pour veiller à l'efficacité et la reproductibilité des applications, les implémentations MPI disposent le plus souvent d'un support de placement guidé par l'utilisateur. Dans un souci de portabilité, l'équipe d'OPEN MPI à mis au point la bibliothèque PLPA [10] (*Portable Linux Processor Affinity*), destinée à uniformiser les méthodes de placement. Elle masque ainsi les évolutions des interfaces existantes et observées sous différentes versions de LINUX.

hwloc
Initialement développée dans l'équipe RUNTIME, la bibliothèque `hwloc` [33] (*Hardware Locality*) offre une abstraction de la topologie matérielle des machines qui intègre flots d'exécution, processeurs, cœurs, puces, nœuds NUMA et la hiérarchie de cache. Elle fournit un ensemble de fonctions permettant leur consultation et leur exploitation, ainsi qu'un outil (`lstopo`) permettant la visualisation des données récoltées. La Figure 2.11 donne un exemple de sortie graphique obtenue à l'aide de `lstopo`. Le niveau de topologie exposé et ses multiples fonctionnalités (placement sur une unité, sur liste d'entités sous un cache commun, au sein d'un même nœud, d'une même puce, etc.) en font une interface largement plébiscitée. Elle a ainsi été incluse dans OPEN MPI en remplacement de la bibliothèque PLPA et bénéficie à de nombreux projets (MPICH2, MVAPICH, DAGuE, PLASMA, etc.).

2.3.2.2 Vers une meilleure collaboration des tâches : des supports exécutifs pour exploiter les affinités sur machines hiérarchiques

Pour exploiter au mieux les architectures modernes et tenir compte des effets de cache et de la localité des données, des travaux s'emploient à exprimer les affinités entre les tâches de sorte à accorder leur placement à la structure topologique matérielle.

Les travaux de Fedorova *et al.* [20, 21] proposent des stratégies d'ordonnancement adaptées pour le partage de cache sur puce multicœurs. Elles assurent un partage équitable des caches entre les différents threads et applications en détectant les contentions sur les caches partagés et en adaptant les quantums de temps accordés à chacun.

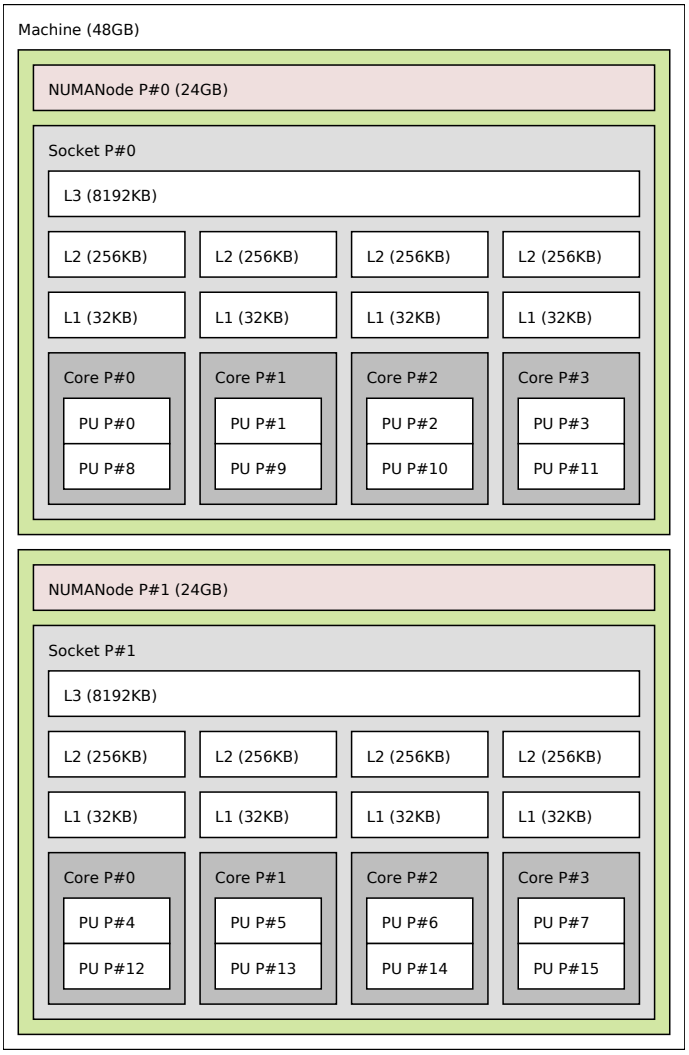


FIGURE 2.11 – *Exemple de sortie graphique de* `lstopo` *décrivant un bi-processeur Intel Xeon quadri-cœurs Nehalem hyperthreadés.*

L'ordonnancement des threads de la bibliothèque MARCEL de la pile logicielle PM2 est affiné grâce à la plateforme *BubbleSched* [103, 102] développée par Samuel Thibault. Elle propose de grouper des threads au sein de structures logiques appelées *bulles* qui expriment des relations d'affinités telles que le partage de données, la participation à des opérations collectives, ou toutes autres relations nécessitant un placement particulier. L'encapsulation dans les bulles permet ainsi un ordonnancement structuré des équipes de threads, initié par diverses stratégies d'ordonnancement en charge de la projection des bulles sur une abstraction de la topologie exprimée grâce à la bibliothèque `hwloc` (Section 2.3.2.1). Ces stratégies répondent à différents besoins applicatifs. Elles veillent par exemple à l'équilibrage de charge, à l'affinité entre les tâches sous un même cache partagé ou encore à la localité des données. La plateforme intègre un support pour l'élaboration de nouvelles propositions d'ordonnancement. Sylvain Jeuland a par exemple proposé une stratégie d'ordonnancement attirant les groupes de threads près des données manipulées favorisant la lo-

calité des données sur architectures NUMA [104].

Dans le cadre de programmes OpenMP, le support ForestGomp [106, 105] proposé par François Broquedis étend l'utilisation des bulles *BubbleSched* pour grouper les threads issus d'une même structure de boucle, assurant ainsi l'affinité matérielle des tâches collaborantes. Le potentiel de parallélisme imbriqué des programmes OpenMP est ainsi harmonieusement agencé sur les topologies hiérarchiques. Le maintien de la projection fine de la structure du parallélisme sur la topologie matérielle est assuré au besoin par de la migration de données et du vol de travail hiérarchique.

Song *et al.* [100, 101] proposent un ordonnancement dirigé par des informations collectées dynamiquement (*feedback*) pour favoriser la localité spatiale et temporelle des programmes sur des topologies mémoire hiérarchiques. L'approche repose sur un outil d'instrumentalisation permettant d'obtenir et d'analyser une trace mémoire pour chaque thread. Il extrait la nature des partages mémoire entre les threads et les traduit dynamiquement en un *graphe d'affinités*. Ce graphe est projeté sur une abstraction de la topologie mémoire et partitionné en sous-graphe définissant des groupes de threads par affinité mémoire. Le calcul d'un ordonnancement adapté sur le matériel disponible est alors stocké dans un fichier pour les exécutions ultérieures, et fait ses preuves sur divers résultats applicatifs. Dans la dernière version du projet, les auteurs adaptent leur algorithme de partitionnement pour supporter les graphes orientés acycliques (DAGs). Grâce à ce perfectionnement, ils offrent ainsi la possibilité d'intégrer les dépendances entres les tâches à leur modèle de calcul d'ordonnancement.

Pour ajuster le placement des données sur machine NUMA, des contributions recalculent leur distribution à partir de traces ou proposent de nouveaux outils de gestion mémoire et de migration de pages. Marathe et Mueller [86] s'appuient par exemple sur des compteurs de performances pour extraire une trace approximative des accès mémoire effectués depuis chaque processeur. L'exécution de la première itération de l'application est utilisée pour générer une trace caractéristique de l'exécution, permettant de dégager un placement NUMA initial efficace, invoqué par *first-touch* pour les exécutions suivantes.

Le principe de *next-touch* est utilisé pour marquer une page qui devra être migrée près du prochain thread qui y accédera. Cette technique permet à un programmeur de placer les pages depuis l'application sans pour autant connaître l'emplacement NUMA d'une tâche, et rectifie une allocation des pages simpliste. Plusieurs implémentations de *next-touch* ont ainsi été développées, par exemple sur système Solaris [88], ou encore sous Linux dans le cadre d'affinités threads/données pour programmes OPENMP [89]. L'interface utilisateur MaMI [85] offre une gestion de la mémoire avec migration explicite ou par *next-touch*. Elle s'appuie sur la détection automatique de l'architecture sous-jacente et met à disposition des utilisateurs des statistiques intéressantes sur l'empreinte mémoire de chaque nœud.

2.3.3 Des conséquences de la topologie sur les communications réseau

La topologie des architectures n'a pas seulement un impact sur la collaboration des tâches présentes sur la machine, mais a également une incidence sur les performances des transferts entre différents nœuds de calcul. Les aspects NUMA sont ainsi connus pour influencer les performances des com-

munications réseau. Sur les machines actuelles, les contrôleurs d'entrées-sorties sont généralement directement connectés aux processeurs par un lien d'interconnexion. Les périphériques réseau se trouvent ainsi plus proches de certains processeurs et nœuds NUMA que d'autres. Les architectures NUMA exhibent ainsi des latences de réception depuis une carte réseau fonction de la localité du processeur qui effectue la réception. Cette caractéristique fait généralement l'objet d'attentions particulières dans le cadre de *microbenchmarks* réseau, pour lesquels des outils de placement tels que `numactl` sont employés pour placer processus et données au plus proche du périphérique utilisé, et garantir des performances optimales et reproductibles. Il existe quelques (rares) supports pour les affinités NUMA dans les bibliothèques de communications. La bibliothèque MX [54] peut par exemple transmettre à l'application le nœud NUMA sur lequel est connectée l'interface réseau.

L'augmentation du nombre d'unités de calcul par nœud engendre la multiplication des accès concurrents aux interfaces réseau, qui peuvent devenir des points de contention. Narayanaswamy *et al.* [38] étudient le comportement du partage des ressources réseau dans les machines multicœurs et dégagent deux comportements possibles. Dans le premier, l'augmentation du pourcentage de communications internes au nœud par rapport aux communications sur le réseau compense les effets des contentions sur les cartes et ne pénalise pas les performances applicatives globales. Dans le second, le partage des ressources crée des contentions sur les périphériques réseau qui détériorent les performances. Il résulte de cette étude que si le facteur d'impact du partage de ressources entre cœurs (caches, mémoire, etc.) est généralement dominant, le partage des ressources réseau peut malgré tout influer sur les performances. Les auteurs observent ainsi le potentiel de différentes stratégies de placement de processus et mettent en évidence la nécessité d'une répartition choisie en accord avec le schéma des communications, pour valoriser les communications indépendantes des réseaux et réduire les contentions sur les cartes.

Pour offrir un meilleur passage à l'échelle et augmenter le débit des communications inter-nœuds, il n'est pas rare de multiplier le nombre de cartes réseau sur les hôtes. Cette stratégie complexifie cependant les problématiques de placement alors que les cartes peuvent êtres attachées à des nœuds NUMA différents. Le problème se pose ainsi de savoir comment distribuer les tâches et les données avec plusieurs bassins de localité réseau, ou comment exploiter les cartes en fonction de leur placement.

Pour préciser l'importance que peut prendre le placement sur les communications réseau, la Figure 2.12 présente les résultats d'un ping-pong *multirail* (utilisant simultanément plusieurs cartes réseau) entre deux machines NUMA. Ce test est réalisé en forçant le placement des tâches et données, à proximité des cartes (courbe rouge), ou au contraire sur un nœud NUMA distant (courbe verte). Plus de détails seront explicités en Section 4.1.3. Nous nous attacherons simplement ici à regarder l'effet sur le débit d'un placement distant du processus de communication, par rapport à un placement NUMA local aux cartes. Alors que pour un bon placement, le débit peu atteindre 1900 Mo/s, il est borné à moins de 1200 Mo/s dans le cas d'un mauvais placement. La dégradation observée est ici de près de 40% ! Un tel résultat prouve que l'influence du placement des tâches sur les performances des communications ne peut être négligée.

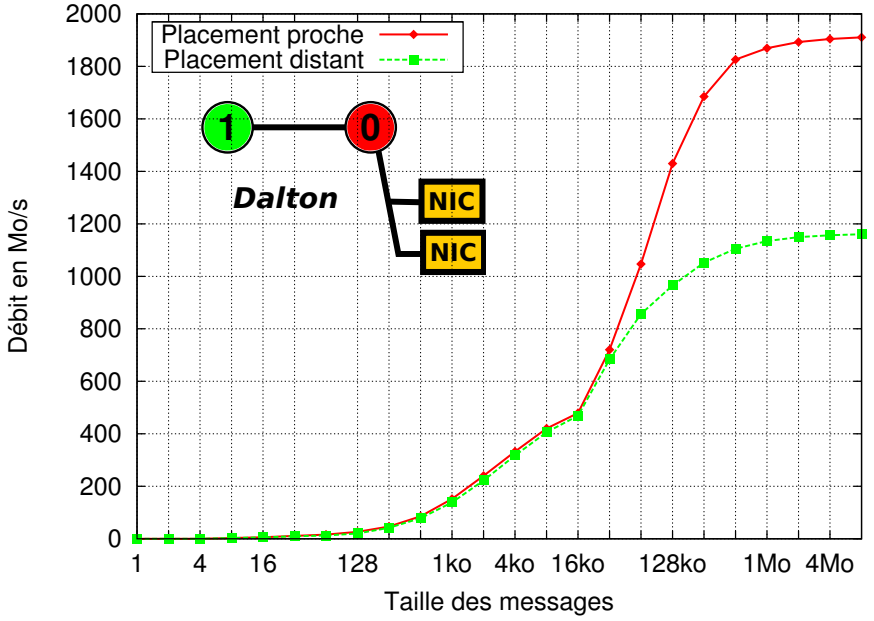


FIGURE 2.12 – *Débits de ping-pong multirails sur la plateforme* NEWMADELEINE *en fonction du placement sur les machines Dalton (détaillées en Annexe A.1).*

2.4 Bilan

L'émergence des architectures parallèles a conduit à la création de plusieurs modèles de programmation, d'interfaces et de standards dédiés. L'évolution des architectures aboutit aujourd'hui à une sévère hiérarchisation des ressources dont les spécificités ont un impact indéniable sur les performances des applications parallèles. Exploiter la quintessence de ces architectures complexes passe par la maîtrise des affinités matérielles créées par l'organisation structurelle des ressources. Les développeurs se voient ainsi contraints d'accorder au mieux le schéma des applications parallèles aux topologies matérielles employées, avec un respect des affinités entre les tâches et les données pour espérer obtenir de bonnes performances. Divers outils de détection de la topologie des architectures et de placement sont ainsi proposés pour leur permettre d'assurer une gestion plus ou moins explicite des affinités au sein même des applications.

L'adéquation entre la structure applicative et la structure matérielle est un prérequis de performance. Malheureusement, les applications (voire leurs développeurs !) disposent rarement d'informations sur leur propre schéma, et de telles indications sont difficilement extraites par les compilateurs. De plus, la responsabilité de projeter intelligemment les applications sur la topologie ne peut être confiée au programmeur qu'au prix d'investissements énormes et souvent au détriment de la portabilité. Des travaux s'orientent ainsi vers une prise en charge de la distribution des tâches et des données au niveau des supports exécutifs. Pour améliorer la collaboration des tâches, ils explorent des ordonnancements guidés par une analyse des dépendances, issue de traces ou de suppositions sur le déroulement de l'application. Pour pouvoir développer le potentiel de gestion des affinités des supports exécutifs, des réflexions sont également menées sur l'évolution des in-

terfaces de programmation dédiées aux architectures parallèles [6].

La multiplication des unités de calcul valorise l'impact des nœuds sur les performances des calculs effectués sur les clusters. La topologie hiérarchique des architectures contemporaines se traduit par autant de niveaux supplémentaires à prendre en considération dans la gestion des affinités sur les grappes. Ces structures interviennent sur la collaboration des processus au sein même des nœuds mais également sur les performances des communications réseau, influencées par la localité des cartes. En l'absence d'accord sur un modèle de programmation réellement adapté au modèle hybride de ces structures, le standard MPI tient lieu de favori dans l'exploitation des grappes tant pour l'échange de données entre les nœuds, qu'au sein même de ceux-ci. Les bibliothèques MPI ont été développées pour exploiter des grappes de machines multiprocesseurs plates, et disposent de multiples stratégies de transferts et d'algorithmes de communication. Pour rester compétitives, elles doivent évoluer et intégrer une prise en compte de la topologie hiérarchique intrinsèque aux architectures contemporaines qui est aujourd'hui un ultimatum d'efficacité.

Dans ce contexte, nous allons étudier les efforts qui peuvent être menés au sein des bibliothèques de communications afin de les adapter aux contraintes de localités inhérentes aux aspects hiérarchiques des architectures et à l'emplacement physique des périphériques.

6. L'idée serait de permettre aux programmeurs d'apporter à leurs codes des indications sur le schéma applicatif, utilisables au niveau de l'environnement d'exécution pour adapter la projection aux aspects hiérarchiques.

Deuxième partie

Contribution

Chapitre 3

Vers des communications qui s'adaptent à la topologie

Sommaire

3.1 Communication et placement **53**
3.2 Affiner les communications sur les grappes à architecture moderne **55**
3.3 Ajuster le placement des processus sur les structures matérielles **56**
3.4 Discussion **57**

Comme nous l'avons vu au chapitre précédent, le placement des tâches a un impact crucial sur les performances. Cette propriété intervient que l'on considère la structure hiérarchique d'un nœud de calcul ou celle d'une grappe, pour laquelle l'efficacité est liée à celle des communications et à l'utilisation de liens réseau plus ou moins lents. L'obtention de bonnes performances lors de l'exécution d'une application parallèle sur un cluster, dépend ainsi non seulement des performances brutes des transferts intra et inter-nœuds, mais aussi de l'adéquation entre le schéma de communication de l'application et la distribution des tâches sur la structure matérielle sous-jacente.

3.1 Communication et placement

Pendant longtemps, l'utilisation de petits nœuds de calcul SMP assurait des communications intra-nœud homogènes de coûts négligeables en comparaison avec ceux des transferts réseau. Les efforts d'optimisation étaient alors axés sur une exploitation efficace de la *structure d'interconnexion* des nœuds des clusters. Les chercheurs ont exploré différentes méthodes de placement des processus MPI au travers des grappes. L'idée était d'adapter ce placement à la structure des applications, pour que le minimum de transferts possibles ait lieu entre des nœuds distincts, voire même de tenir compte des distances réseau entre les différents nœuds, pour assurer l'amélioration des performances applicatives et leur reproductibilité. Par exemple, si la structure applicative est caractérisée par de très forts échanges de données entre paires de processus comme illustré par les flèches rouges de la Figure 3.1, la localité des processus induite par le placement (a) sur la

topologie à deux niveaux proposée avantage les performances en comparaison du placement (b) qui nécessite des transferts entre les niveaux distincts.

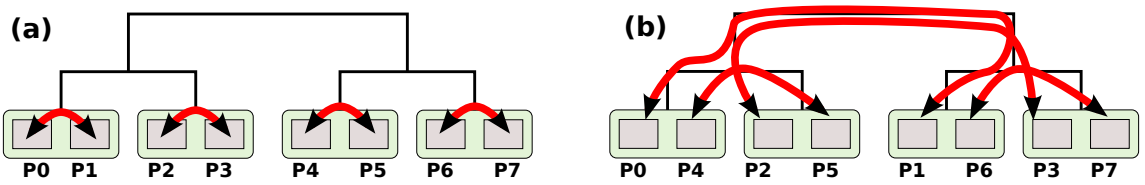


FIGURE 3.1 – *Représentation du trafic interprocessus en fonction du placement des tâches :*
(a) favorisant la localité des transferts applicatifs,
(b) indépendant du schéma de communication applicatif et à l'origine de forts trafics distants.

Les applications MPI disposent de multiples schémas de communication. On distingue notamment les communications point-à-point (effectuées entre deux tâches), et les communications collectives impliquant un ensemble plus large de processus collaborants. Les efforts de placement se sont combinés avec l'élaboration d'algorithmes efficaces [1] pour les opérations collectives, réduisant le nombre d'envois nécessaires et dont la projection sur la structure de la grappe avantage la localité des transferts.

La complexification croissante des nœuds et l'augmentation du nombre d'unités de calcul disponibles sur chacun d'entre eux ouvrent une nouvelle dimension de difficulté dans l'exploitation des grappes de calcul. En effet, l'augmentation du nombre de cœurs s'accompagne de la multiplication du nombre de tâches par nœuds, et donc de l'importance des communications intra-nœud. Même si le coût des communications sur le réseau reste supérieur à celui des transferts internes à un hôte, l'impact des communications intra-nœud sur les performances globales ne peut être négligé.

Une bonne exploitation des clusters contemporains résulte ainsi d'une gestion efficace des transferts intra-nœud et inter-nœuds ainsi que du placement des tâches, et implique un large spectre d'efforts à fournir. Si le placement des tâches impacte les performances, les schémas de communication et algorithmes employés, ainsi que les mécanismes de transfert utilisés jouent également un rôle majeur. Pour prendre en compte les affinités entre la structure matérielle et les communications, il est possible d'adapter le placement des processus à la topologie pour favoriser les échanges les plus locaux. Une méthode orthogonale consiste à sélectionner les stratégies de communication les plus efficaces en fonction d'une répartition prédéfinie des processus sur l'architecture cible.

Les échanges invoqués se divisent en deux groupes, intra-nœud et inter-nœuds. Du point de vue interne à un hôte, on distingue les transferts entre deux tâches locales à la machine (soumis aux effets de cache, NUMA, etc.), et les transferts vers les périphériques réseau (sujet à la localité des cartes comme en témoigne l'expérience présentée en Section 2.3.3). Pour comprendre comment il est possible d'intégrer la complexité des architectures modernes des nœuds de calcul dans l'exploitation des grappes, nous présentons dans ce chapitre les efforts allant dans ce sens, au niveau des stratégies de communication et algorithmes employés, puis au niveau des efforts de placement.

1. basés sur des arbres de diffusion, des groupements de messages, etc.

3.2 Affiner les communications sur les grappes à architecture moderne

Pour assurer un premier niveau de portabilité des performances, des propositions utilisent conjointement des algorithmes complémentaires pour ajuster au mieux les communications selon les critères de transfert et les plateformes utilisées. Les travaux de Vadhiyar, Fagg et Dongarra [157, 174] proposent une sélection automatique des algorithmes pour maximiser les performances sur une plateforme cible, basée sur une expérimentation préalable. Plusieurs contributions produisent des *algorithmes hybrides* dont les permutations sont indiquées par les caractéristiques des communications [169, 170, 176, 172, 177]. Thakur *et al.* [176, 177] ont ainsi amélioré les performances de leurs algorithmes dans MPICH en sélectionnant les stratégies utilisées sur la taille des messages et le nombre de processus. Par exemple, leur implémentation de l'opération de diffusion (*broadcast*) s'appuie sur l'utilisation d'un arbre binomial pour des messages de taille inférieure à 12 ko, ou lorsque le nombre de processus n'excède pas 8. Dans les autres cas, cet algorithme apparait moins performant que la méthode développée par Van de Geijn *et al.* [169] qui est alors employée. Pješivac-Grbović et Graham [175] envisagent quant à eux une sélection des algorithmes à partir de *quadtrees* créés depuis les performances relevées des différents algorithmes.

D'un point de vue général, l'ensemble des implémentations compétitives s'appuient aujourd'hui sur des algorithmes hybrides ajustés selon les critères des transferts. Des travaux calibrent le choix des algorithmes et leur transition sur des architectures particulières, notamment pour des machines telles que la *Blue Gene* [197]. Pour répondre à la diversité des plateformes de calcul, les implémentations les plus portables s'appuient sur une configuration de paramètres, explicitée par les utilisateurs ou automatique (par le biais d'*autotuning*) [77, 75, 76, 74]. Certaines définissent des heuristiques générales, efficaces dans *la plupart des cas*, comme par exemple le module OPEN MPI *Tuned* [71]. Initialement conçu sur un cluster de processeurs AMD64 interconnectés par Gigabit Ethernet, ce module utilise des heuristiques définissant comment découper et pipeliner les messages, et qui s'accordent plutôt bien avec un grand nombre de plateformes. En l'absence d'indications contradictoires de la part de l'utilisateur, il est ainsi utilisé comme composant par défaut d'OPEN MPI.

La conception d'algorithmes efficaces doit désormais prendre en compte les caractéristiques spécifiques aux architectures modernes. Nous présentons ici les travaux récents qui proposent des algorithmes basés sur l'organisation des architectures multicœurs.

Le trafic mémoire, la concurrence de cache et les synchronisations sont les barrières limitant le passage à l'échelle et les performances des opérations collectives en mémoire partagée. Les travaux de Graham *et al.* [180] implémentent ainsi des algorithmes (broadcast, reduce et allreduce) efficaces qui réduisent le trafic mémoire en limitant le nombres d'écrivains sur un segment mémoire donné et équilibrent les synchronisations et les coûts d'accès mémoire. En plus de réduire le trafic au niveau des puces, ces algorithmes permettent d'exploiter les caches partagés et réduisent les échanges inter-sockets. Tu *et al.* [195] reprennent quant à eux les algorithmes multiniveaux de MPICH2 et proposent une subdivision des messages en segments réordonnés pour profiter des communications interprocessus via caches partagés et bénéficier de la localité temporelle des données comme suggéré dans [36] et [199].

Certaines contributions intègrent également à leurs stratégies la prise en compte du niveau de topologie NUMA. Mamidala *et al.* [178] appuient par exemple leur implémentation sur les caractéristiques des architectures multicœurs INTEL *Clovertown* et AMD OPTERON pour proposer des algorithmes d'opérations collectives profitant de l'utilisation des caches partagés et des propriétés bidirectionnelles des liens HYPERTRANSPORT. Kandalla *et al.* ajoutent un niveau de hiérarchie supplémentaire à leur approche *multi-leader* [179] en définissant plusieurs *processus maîtres* par nœud. Chaque puce OPTERON de leur plateforme, (et donc chaque nœud NUMA), dispose ainsi de son leader local. L'adaptation des algorithmes de collectives à la structure interne des architectures multicœurs et NUMA suscite également l'intérêt des constructeurs. Cette piste est par exemple explorée par l'entreprise Bull pour ses propres bibliothèques [13].

La recherche pour l'optimisation des stratégies et schémas de communication constitue un noyau solide depuis longtemps. Il est aujourd'hui renforcé par la complexification des architectures modernes qui engage les chercheurs vers de nouvelles pistes de développement. Alors que l'utilisation efficace du matériel réseau semble relativement aboutie, les efforts se concentrent sur le choix des stratégies et algorithmes de communication employés à l'intérieur des nœuds multicœurs.

3.3 Ajuster le placement des processus sur les structures matérielles

Le placement des processus des applications parallèles sur les cœurs des nœuds de calcul a un impact indéniable sur les performances des applications, mais est également un prérequis à leur reproductibilité. Les bibliothèques de communication intègrent donc généralement des fonctionnalités de placement des processus, éventuellement paramétrables.

L'idéal serait d'obtenir une distribution des tâches en accord avec les topologies des architectures et des schémas de communication invoqués par les applications dans leur ensemble. La diversité des architectures et des applications oblige à avoir une bonne connaissance des machines cibles et des structures applicatives pour espérer planifier une répartition efficace des tâches. De plus, même en disposant de telles informations, les multiples spécificités des architectures (présentées en Section 2.3) rendent quasiment impossible une prédiction fine des performances par un utilisateur.

Dans ses travaux, Jesper Larsson Träff [143] montre l'intérêt de l'utilisation de la théorie des graphes pour optimiser le placement des processus MPI. La répartition des tâches fait l'objet de nombreux efforts [137, 138, 139, 140], mais la plupart des solutions requièrent un investissement des utilisateurs ou ne répondent pas toujours aux critères topologiques des architectures multicœurs actuelles. Des pistes récentes s'orientent donc vers une extraction fine de graphes de communication et de topologie issus de profilage, pour guider les stratégies vers un placement adapté aux topologies hiérarchiques modernes.

Parmi les projets les plus aboutis, la boîte à outils MPIPP (*MPI Process Placement toolset*) [133] inclut des outils de profilage des communications et de détection de topologie réseau des grappes. Les informations récoltées peuvent ainsi êtres utilisées pour déterminer la répartition des processus sans intervention de l'utilisateur ou de connaissance du schéma de communication. L'approche OPP (*Optimized Process Placement*) [136] permet d'affiner le graphe de communication de MPIPP, basé sur les échanges point-à-point, en décomposant les opérations collectives en séries

de transferts point-à-point inclus dans l'analyse. Adapté pour des clusters de machines SMPs, le niveau de topologie exprimé par MPIPP n'est cependant pas suffisamment profond pour exploiter la hiérarchie des multicœurs.

Le placement des processus en fonction de la topologie de la machine cible à l'aide de la théorie des graphes est également étudié par les constructeurs tels que HEWLETT-PACKARD [134] et IBM [135]. Les bibliothèques de HP [134] extraient ainsi d'une première exécution le profil de communication de l'application et adaptent le placement en fonction des distances de communication évaluées. Pour mettre en œuvre leurs politiques de placement, les travaux d'IBM [135] définissent des graphes de distance matérielle des communications, en fonction des paramètres de connectivité (latence et débit entre paires de tâches communicantes), ainsi que des schémas de communication applicatifs, extraits du nombre et de la taille des données échangées entre les paires.

L'évaluation de Mercier et Clet-Ortega [131] témoigne de l'importance de la politique de placement sur les performances globales des applications exécutées sur machines hiérarchiques. Elle démontre que l'augmentation des communications intra-nœud dégrade significativement les performances lorsque l'utilisation des caches n'est pas faite de façon réfléchie. Pour y remédier, ils proposent une stratégie de placement appuyée sur une analyse du modèle de communication et la topologie matérielle, en effectuant un plongement du graphe de communication dans celui de la topologie à l'aide du logiciel de partitionnement de graphes SCOTCH [141]. L'algorithme utilisé, TREEMATCH [132], bénéficie d'un arbre d'éléments matériels précis dérivé de la bibliothèque `hwloc` [33] (présentée en Section 2.3.2.1). En utilisant un modèle de communication axé sur la quantité de données échangées entre les paires de processus, il effectue une répartition des processus MPI au travers de la machine particulièrement efficace si le schéma de communication de l'application est lui-même hiérarchiquement structuré.

Si les utilisateurs sont rarement conscients de la structure des machines, certains ont de très bonnes notions du schéma applicatif de leurs algorithmes. Une proposition récente envisage d'offrir aux utilisateurs la possibilité d'expliciter un graphe de communication représentatif de l'application, et se charge de réordonner les processus sur l'architecture matérielle [61].

Pour orchestrer une bonne adéquation entre la topologie des applications et le placement effectué par les bibliothèques de communication, une autre piste consisterait à faire transiter des indications entre la couche logicielle et les intergiciels. Pour permettre un tel procédé, des groupes de travail tels que le *MPI-3 Hybrid Working Group* étudient l'évolution potentielle des standards pour y inclure l'ajout d'informations dans cet objectif.

3.4 Discussion

Les performances des applications parallèles sont tributaires des performances des transferts de données entre les tâches contribuant au travail. Sur les clusters de calcul, ces transferts interviennent au niveau des communications réseau, effectuées entre les nœuds de calcul, et au sein même des nœuds multiprocesseurs et multicœurs.
Longtemps, l'amélioration des échanges a été portée par la réduction du nombre et des coûts de communication réseau. De nombreux efforts ont ainsi été menés pour favoriser une exploitation

efficace des réseaux haute performance dont les technologies n'ont cessé d'être perfectionnées par les constructeurs. Côté algorithmique, les chercheurs ont travaillé au perfectionnement des schémas d'échange, pour minimiser le nombre d'envois au travers des réseaux au profit des échanges locaux aux nœuds. La recherche d'une projection fine de l'application sur la structure des grappes est devenue un prérequis à une exploitation efficace de celles-ci. Elle a conduit les chercheurs à penser la répartition des tâches sur la topologie de grappe en accord avec les affinités logicielles entre les tâches, et à définir des algorithmes de communication efficaces sur cette structure. L'utilisation d'architectures multiprocesseurs plates a porté un concours entier à la puissance des clusters de calcul, augmentant le nombre d'unités de calcul et permettant de déporter une partie des communications au sein du nœud. Le modèle à mémoire partagée qu'elles possèdent a enrichi les algorithmes de communication de nouveaux protocoles et servi les affinités applicatives.

La généralisation du multicœur a conduit à une hiérarchisation inévitable des structures qui composent les calculateurs modernes et leur topologie interne ajoute une dimension supplémentaire à l'utilisation des grappes. En effet, les effets NUMA, la hiérarchie de cache et la localité des périphériques impactent largement les performances des transferts intra-nœud et réseau. De plus l'augmentation du nombre d'unités de calcul sur chaque nœud revalorise l'importance des communications intra-nœud victimes de ces effets qui peuvent alors être un frein sévère aux performances. Les problématiques de placement sur les grappes ne peuvent plus se résumer à une distribution sur les nœuds du cluster, mais bel et bien au niveau de chaque unité de calcul imbriquée dans une hiérarchie de ressources complexe. La topologie des grappes en est ainsi enrichie d'autant de niveaux hiérarchiques supplémentaires à prendre en considération.

Contribution

La prise en charge de ces niveaux structurels est un problème d'autant plus complexe que la variété des architectures et des technologies employées ne permet pas de généraliser leur effets sur les performances. C'est dans ce contexte que s'inscrivent les travaux de cette thèse. Ils ont pour objectif d'explorer des pistes permettant une prise en compte de la topologie interne des calculateurs contemporains pour le calcul haute performance sur grappe. Ils intègrent une évaluation de certaines caractéristiques propres aux architectures modernes et la recherche de méthodes portables pour l'amélioration des performances dans le contexte du placement des tâches et des mouvements de données entre celles-ci.

Les contributions possibles pour l'exploitation des grappes de machines hiérarchiques peuvent être faites à plusieurs niveaux dans la pile logicielle et peuvent être découpées selon deux axes. Le premier consiste à adapter les stratégies de communication en considérant un placement des tâches prédéfini. Le second s'articule autour de la recherche d'un placement des tâches optimal en fonction des schémas des communications mis en œuvre. La décomposition de ces axes dans les problématiques de communication inter et intra-nœud, aboutit à quatre champs d'intervention illustrés par la Figure 3.2 :

- l'**optimisation des transferts intra-nœud** en accord avec la distribution des tâches,
- l'**optimisation des transferts réseau (inter-nœuds)** en accord avec la distribution des tâches,
- l'**adaptation du placement** des tâches relatif aux communications intra-nœud,
- l'**adaptation du placement** des tâches relatif aux transferts réseau.

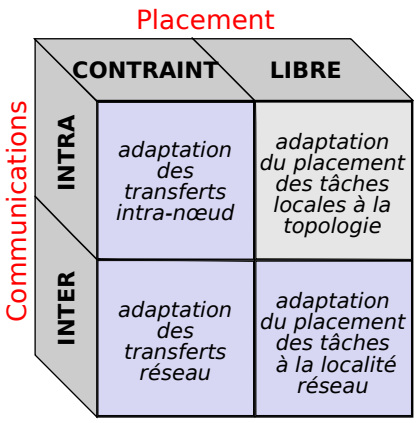


FIGURE 3.2 – *Champs d'intervention pour une prise en compte de la hiérarchie des nœuds de calcul considérant les découpages :*
– adaptation des stratégies de communication ou adaptation du placement des tâches,
– communication intra-nœud ou communication réseau.

Une solution idéale serait d'intervenir à tous les niveaux. Il s'agit bien sûr d'un problème terriblement difficile [2], alors que de telles interventions doivent être judicieusement harmonisées avec les problèmes concurrents d'équilibrage de charge, d'ordonnancement et de portabilité. Ces problématiques sont également tributaires d'informations relatives à la topologie des architectures et aux patrons de communication applicatifs qui dépendent des problèmes traités et dont la détection peut être délicate.

Aucun calcul scientifique d'ampleur ne peut être structuré de façon similaire aux topologies matérielles des grappes composées de machines hiérarchiques, faisant d'une adéquation *parfaite* entre la structure des applications et celle des calculateurs, une utopie. La recherche d'une projection *optimale* est concevable mais ne peut se faire qu'au détriment de la portabilité alors que l'évolution et le renouvellement des calculateurs employés s'accompagnent de nouvelles organisations. Elle impliquerait de surcroît un investissement incroyable de la part des programmeurs, seuls capables d'étudier en détails les structures applicatives et matérielles pour en extraire une adéquation idéale. Entre une projection naïve et simpliste et une projection optimale, il existe cependant un vaste terrain de perfectionnements possibles. Un bon compromis entre adaptation aux structures hiérarchiques et minimisation de l'investissement des programmeurs consiste à déléguer la charge des aspects de topologie matérielle aux bibliothèques et supports exécutifs. Ceux-ci devraient ainsi être capable d'adapter dynamiquement l'exécution des programmes à la structure matérielle, assurant une amélioration de performances tout en garantissant leur portabilité.

Alors que la piste d'une distribution générale des tâches applicatives sur l'architecture de grappe est déjà l'objectif de plusieurs travaux (voir Section 3.3), les recherches menées dans le cadre de cette thèse s'orientent sur les trois autres axes présentés en Figure 3.2 (colorés en bleu clair). Ces pistes sont exploitables à un niveau logiciel bas et leur traitement peut être intégré au sein des

2. Pouvant être catégorié de NP-difficile.

bibliothèques de communication sans trop endommager les efforts applicatifs.

Pour des raisons d'affinités logiques, les propositions relatives aux pistes pour les communications réseau sont présentées successivement dans ce document. Comme le suggère la Figure 2.12 vue en Section 2.3.3, l'emplacement des périphériques réseau sur la topologie interne des calculateurs a un impact sur les performances des transferts réseau. L'évaluation de cette caractéristique sur différents clusters nous a conduits à intégrer les contraintes de localité réseau dans les bibliothèques et à explorer le potentiel d'adaptation du placement des tâches. Nous nous sommes ensuite attachés à la proposition orthogonale qui consiste à adapter les méthodes de communication employées dans le cadre d'applications MPI, pour lesquelles la répartition des tâches est généralement prédéterminée.

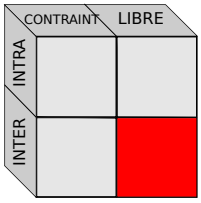
Si la topologie joue un rôle sur les performances réseau, son implication sur les échanges entre deux tâches locales à une machine n'en est pas moindre. Les bibliothèques de communication MPI bénéficient comme nous l'avons vu, de multiples stratégies de transfert intra-nœud utilisées conjointement (Section 2.1.2.1). Chacune d'entre elles exploite des ressources matérielles différentes qui déterminent les performances des transferts effectués. Leur bénéfice est ainsi étroitement lié à la topologie de la machine exploitée. Considérant ces stratégies, l'obtention de performances optimales demande ainsi de savoir déterminer la méthode la plus efficace en fonction du contexte de communication (schéma point-à-point ou collectif, taille des messages, etc) et des caractéristiques matérielles intervenantes. Pour répondre à cette exigence, nous avons évalué les performances des stratégies de transfert sur plusieurs architectures, dans le but d'harmoniser de façon portable leur utilisation avec les caractéristiques matérielles propres à la topologie sous-jacente. Cette proposition présentée en Chapitre 6 s'insère dans le champ d'intervention relatif à l'adaptation des stratégies de communication intra-nœud pour un placement contraint des tâches.

Chapitre 4

Des effets NUIOA à l'adaptation du placement pour les communications réseau

Sommaire

4.1 Évaluation de l'impact des effets NUIOA sur les communications 62
4.1.1 Plateforme de test 62
4.1.2 Effets NUIOA sur les transferts locaux 64
4.1.3 Répercussion des effets NUIOA sur les communications applicatives . 67
4.1.4 Spectre d'impact NUIOA 76
4.1.5 Bilan 80
4.2 Implémentation d'une solution de placement automatique 81
4.2.1 Trouver le nœud le plus proche de chaque carte réseau 82
4.2.2 Mise en œuvre dans NEWMADELEINE 82
4.2.3 Cas des communications multirails 83
4.3 Bilan 84



Dans le domaine des communications sur réseaux rapides au sein des grappes de calcul, il est connu que les tests de performance réseau doivent être lancés avec un placement NUMA contrôlé, pour afficher des performances optimales et reproductibles. En effet, comme nous l'avons remarqué sur les courbes de la Figure 2.12, le transfert de données depuis un nœud distant de l'interface réseau peut dégrader significativement les performances des communications. Ce chapitre présente l'impact de ces effets NUIOA (*Non Uniform Input/Output Access*) sur les performances des communications sur réseaux rapides. Nous cherchons dans un premier temps à quantifier ces effets et à identifier les cas dans lesquels ils s'avèrent significatifs. Nous proposons ensuite une solution de placement automatique des tâches communicantes et des tampons mémoire associés au sein de la bibliothèque de communication NEWMADELEINE pour minimiser les effets NUIOA et maximiser les performances globales des transferts.

4.1 Évaluation de l'impact des effets NUIOA sur les communications

Cette première partie présente une évaluation des effets NUIOA pour différentes architectures et technologies réseau. Pour estimer l'influence du placement des threads et de la mémoire sur la latence et le débit, nous avons réalisé une série de tests réseau basés sur :

– des transferts locaux, entre la carte réseau et la mémoire, permettant de mesurer les effets NUIOA intervenant au niveau de la machine,
– des communications entre deux hôtes (ping-pong) pour évaluer les répercussions de ces caractéristiques au niveau applicatif.

4.1.1 Plateforme de test

Les tests présentés dans cette partie ont été effectués sur différentes architectures que nous présentons succinctement. La topologie et les caractéristiques de l'ensemble des machines utilisées sont détaillées en Annexe A. La première partie de notre plateforme de test (Figure 4.1) est composée d'hôtes dotés de processeurs AMD OPTERON et de diverses interfaces réseau : MYRICOM MYRI-10G, QUADRICS QSNET II et MELLANOX INFINIBAND. Sur les architectures OPTERON, chaque interface est connectée à une puce au travers d'un lien HYPERTRANSPORT, et se trouve ainsi proche d'un nœud NUMA [1] et à distance des autres.

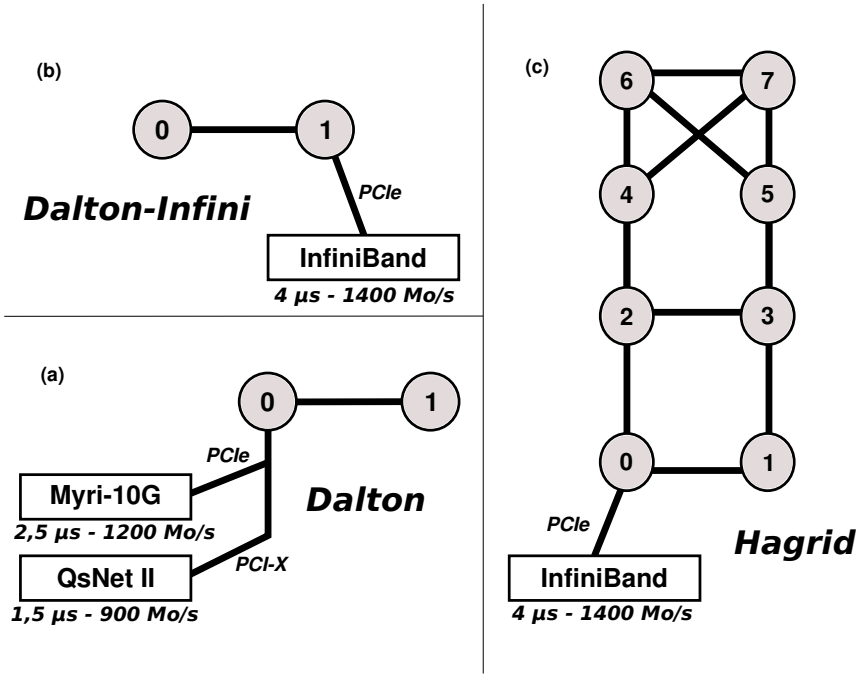


FIGURE 4.1 – *Topologies* NUMA *et réseau de notre plateforme de test* OPTERON.

1. Association d'une puce OPTERON et d'un banc mémoire.

Les machines *Dalton* (Figure 4.1 (a)) sont des architectures NUMA bi-processeurs bi-cœurs AMD OPTERON. Elles disposent de deux interfaces réseau (NICs), MYRI-10G et QSNET II attachées sur le nœud NUMA 0, qui exhibent des latences de 2,5μs et 1,5μs respectivement, et des bandes passantes de 1200 Mo/s et 900 Mo/s. Les machines *Dalton-Infini* représentées Figure 4.1 (b) diffèrent des Dalton par la présence d'une seule interface de technologie INFINIBAND attachée au nœud 1 et dont les performances sont de 4μs de latence et de 1400 Mo/s de bande passante. A ces machines s'ajoute la machine à huit nœuds *Hagrid* dont la topologie NUMA est présentée Figure 4.1 (c). Elle dispose d'une carte INFINIBAND équivalente à celle des Dalton-Infini, attachée sur le nœud 0.

Outre nos différentes architectures OPTERON, notre plateforme de test a été enrichie de serveurs INTEL dédiés à l'étude des accélérateurs graphiques. Parmi eux la machine *Hannibal* (Figure 4.2 (a)) est un bi-processeur INTEL Xeon composé de puces Nehalem X5550 et dotée de trois GPUs (détails en Annexe A.6). Cette plateforme ne dispose normalement pas d'interface de réseau rapide. Pour les besoins de nos travaux, nous avons momentanément remplacé l'un des GPUs disponibles par une carte INFINIBAND Connect-X QDR [2] (Figure 4.2 (b)) qui exhibe une latence de l'ordre de 1,4μs et un débit supérieur à 2500 Mo/s. Nous avons également travaillé sur le cluster *Mirage* composé de machines bi-processeurs hexa-cœurs INTEL Westmere Xeon X5650 (détaillées en Annexe A.5). Chacun de ces hôtes (Figure 4.2 (c)) comprend 3 GPU et une carte INFINIBAND dont la connectivité QDR soutient des performances de 1,8μs de latence et 2300 Mo/s de débit.

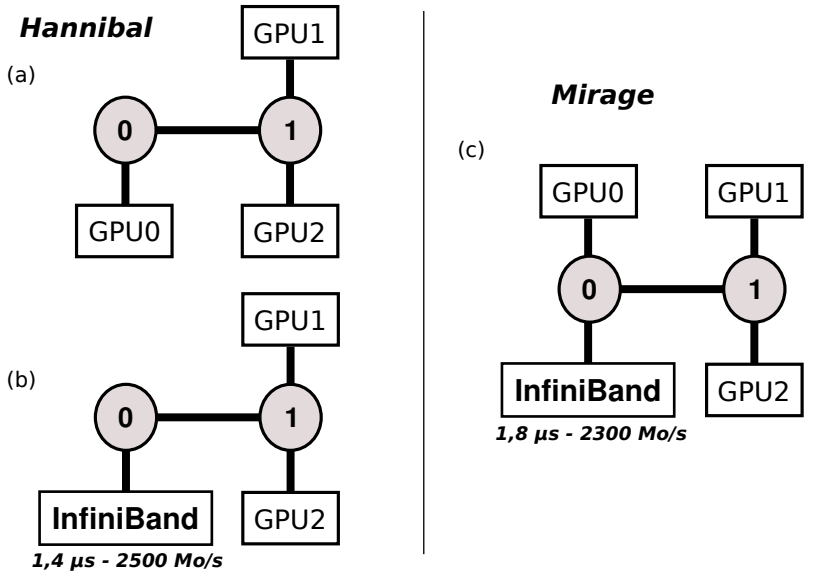


FIGURE 4.2 – *Topologies* NUMA *et réseau de notre plateforme de test Intel.*
(a) Machine Hannibal dotée de 3 GPUs (configuration classique).
(b) Machine Hannibal avec une interface INFINIBAND *(configuration temporaire).*
(c) Machines du cluster Mirage.

2. Quad Data Rate.

4.1.2 Effets NUIOA sur les transferts locaux

Pour jauger l'influence du placement des tâches et de la mémoire sur la latence des accès au périphérique réseau, nous avons mesuré les performances des transferts locaux, entre la mémoire et la carte réseau (Figure 4.3). Nos expériences s'appuient principalement sur des tests constructeurs prenant en charge un mouvement des données bas niveau, épuré de surcoût logiciel (*microbenchmarks* et test de transferts). Les tests sont effectués sur les différentes interfaces de notre plateforme en faisant varier le placement des tâches de communication sur les différents nœuds NUMA. Pour chacun, la tâche communicante et les données sont explicitement placées sur le nœud souhaité.

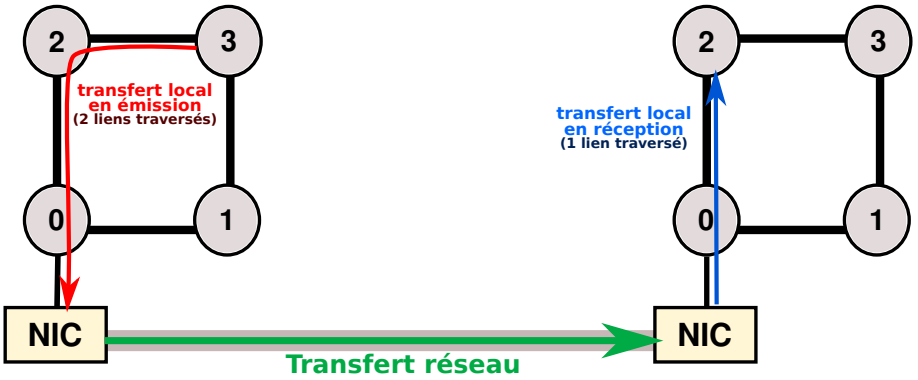


FIGURE 4.3 – *Exemple de transfert de données entre deux hôtes. Les données transitent au travers de deux liens d'interconnexion pour atteindre l'interface réseau côté émetteur, et au travers d'un lien pour atteindre leur destination côté récepteur.*

Effets NUIOA sur la latence des transferts locaux

Pour observer l'impact du placement sur la latence des communications nous étudions le transfert de petites requêtes. Les tests mesurent le délai entre l'envoi d'une requête vide à l'interface réseau et le moment où l'événement correspondant est notifié à l'application. La requête est traitée par la carte mais ne génère pas de communication réelle, la notification peut être effectuée par PIO ou DMA selon l'interface utilisée. La Table 4.1 présente les résultats de nos expériences en fonction du placement des tâches sur nos machines à deux nœuds OPTERON.

	nœud proche	nœud distant	surcoût
MYRI-10G	1739	1794	**55**
QSNET II	1610	1670	**60**
INFINIBAND	464	559	**95**

TABLE 4.1 – *Impact du placement* NUMA *sur la latence (en nanosecondes) de requêtes de transfert avec différentes technologies réseau, sur des machines à 2 nœuds.*

Les résultats pour le réseau MYRI-10G ont été obtenus à partir du test `mx_piobench` fourni par le constructeur de ce réseau (MYRICOM). Ce test envoie des requêtes de 64 octets en PIO suivies de complétions par DMA. Le test en mode DMA `qsnet2_dmatest`, mis à disposition par le

constructeur QUADRICS, donne des informations de latence lorsqu'il est utilisé avec des tailles de message très petites, et nous permet d'évaluer l'impact du placement sur la latence pour cette interface. Enfin, le réseau INFINIBAND ne disposant pas de test constructeur, nous avons évalué les variations de latence à partir de notre propre test de PIO, qui effectue une écriture (*PIO write*) puis une lecture (*PIO read*) dans la mémoire du périphérique réseau. Ces outils sont spécifiques à chaque interface et leurs résultats n'ont pas vocation à être comparés entre eux. Ils fournissent toutefois des méthodes assez équivalentes nous permettant d'évaluer l'impact NUIOA du placement sur la latence.

Les résultats nous montrent un surcoût entre 55 et 95 nanosecondes pour un aller-retour entre le processeur et l'interface réseau, soit en moyenne près de 40 ns pour la traversée d'un lien HYPERTRANSPORT. Cette expérience a été étendue à notre architecture à huit nœuds *Hagrid*. Les résultats présentés Figure 4.4 confirment que le transfert unidirectionnel augmente d'environ 40 ns pour chaque lien HYPERTRANSPORT supplémentaire traversé.

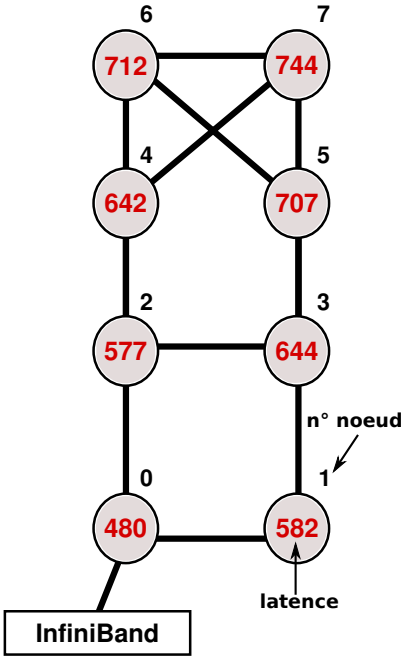


FIGURE 4.4 – *Latences (en nanosecondes) relatives au placement pour un aller-retour PIO entre un processeur et l'interface* INFINIBAND *sur une machine à huit nœuds.*

Nous n'avons pu mener les tests équivalents qui nécessitent des droits super-utilisateurs sur nos plateformes INTEL, dont les processeurs sont interconnectés grâce à la technologie QUICKPATH INTERCONNECT [3] (QPI). Des tests applicatifs simples (ping-pong de messages vides) montrent toutefois un surcoût de latence de l'ordre de 170 ns par lien QPI traversé et laissent présager un surcoût au niveau des transferts locaux de quelques dizaines de nanosecondes par lien.

3. A l'origine d'une décomposition en nœud NUMA pour chaque ensemble *processeur-mémoire*.

Effets NUIOA sur le débit des transferts locaux
L'impact NUIOA sur le débit des transferts locaux est ici estimé en examinant les performances des transferts DMA (adaptés aux données de grande taille utilisées pour évaluer le débit) vers et depuis le périphérique réseau. Nous nous appuyons pour cela sur les tests `mx_dmabench` et `qsnet2_dmabench` fournis par les constructeurs MYRICOM et QUADRICS, qui effectuent des transferts vers et depuis la carte réseau en mode DMA [4].

Le réseau INFINIBAND ne disposant pas de test de DMA dédié, nous n'avons pu restreindre nos expérimentations aux seuls transferts locaux. Les résultats présentés ici pour le réseau INFINIBAND sont ainsi appuyés sur des tests de RDMA (*Remote Direct Memory Access*). Ils impliquent des envois réels au travers du réseau et présentent ainsi un débit potentiellement inférieur à celui attendu pour les seuls transferts DMA sur ce matériel. En faisant varier uniquement les zones mémoires cibles, ou au contraire sources du transfert [5] les résultats obtenus devraient cependant témoigner des variations relatives aux transferts locaux.

		Placement proche	Placement distant	Impact
QSNET II	Lecture	879	879	0 %
	Écriture	925	925	0 %
MYRI-10G	Lecture	1308	1295	- 1 %
	Écriture	1518	1162	- 24 %
INFINIBAND [6](DDR)	Lecture	*1390*	*1390*	*0 %*
	Écriture	*1392*	*1071*	*- 23 %*
Accès mémoire sur la plateforme Dalton	Lecture	2696	1871	- 31 %
	Écriture	4765	2952	- 38 %

TABLE 4.2 – *Impact du placement* NUMA *sur le débit (Mo/s) pour des transferts locaux par DMA de la carte vers la mémoire (écriture), de la mémoire vers la carte (lecture) et des accès mémoire sur des machines* OPTERON *à 2 nœuds* NUMA.

Les résultats mesurés sur les différents processeurs de nos machines OPTERON à deux nœuds sont donnés en Table 4.2. **Pour un placement distant, ils révèlent une baisse de performance d'autant plus significative que le débit est élevé**. Les résultats pour QSNET II ne montrent pas de variation relative au placement tandis que ceux pour INFINIBAND et MYRI-10G témoignent d'une dégradation de débit notable, allant jusqu'à 24%. **Cette perte apparaît en outre asymétrique**, les lectures DMA (transferts de la mémoire vers la carte réseau) ne semblent pas affectées par le placement alors que les écritures (de la carte vers la mémoire de l'hôte) le sont. L'observation des impacts NUMA sur de simples copies mémoire (présentées en fin de tableau) montre des effets concordants de la distance NUMA, avec une dégradation liée au placement qui augmente avec le débit théorique et s'avère plus importante pour une écriture que pour une lecture.

Nous avons étendus ces résultats à nos architectures à deux nœuds INTEL (Table 4.3). Nous observons bien une dégradation d'autant plus élevée que le débit est grand (atteignant ici 40%), mais les

4. Pour ces tests constructeurs, les données ne sont pas réellement transmises au travers du réseau et sont ignorées par l'interface.

5. L'absence d'effet NUIOA sur l'hôte partenaire est assuré avec un placement mémoire local vis à vis de la carte réseau.

6. Basés sur de réelles communications et non pas sur un microbenchmark de transfert DMA.

		Placement proche	Placement distant	Impact
INFINIBAND[6] (QDR)	Lecture	*2966*	*2859*	*- 3,6 %*
(machine *Nehalem* Hannibal)	Écriture	*2340*	*1525*	*- 35 %*
INFINIBAND[6] (QDR)	Lecture	*2232*	*1343*	*- 40 %*
(machine *Westmere* Mirage)	Écriture	*2230*	*1889*	*- 15 %*
Accès mémoire sur la	Lecture	5007	4468	-11 %
plateforme **Mirage**	Écriture	5162	2986	-42 %

TABLE 4.3 – *Impact du placement* NUMA *sur le débit (Mo/s) pour des transferts DMA locaux en lecture et écriture sur des machines* INTEL *à 2 nœuds.*

effets asymétriques présentent de nouveaux aspects. Tout d'abord, la réduction de débit est visible à la fois pour les écritures et les lectures DMA. Nous pensons que cette caractéristique est observable grâce à la très forte bande passante obtenue sur ces architectures et pourrait être présente sur les dernières générations de plateformes OPTERON [7] avec des cartes réseau récentes.

De plus, contrairement à l'ensemble des architectures sur lesquelles nous avons eu l'occasion d'effectuer des tests, les machines *Mirage* présentent une dégradation de débit impactant majoritairement les *lectures* DMA. Ce résultat est d'autant plus surprenant que les écritures mémoire sur cette architecture sont elles, plus affectées par la distance NUMA que les lectures (dernière ligne de la Table 4.3). Il semblerait ainsi que le comportement asymétrique des effets NUIOA sur le débit des transferts locaux soit lié au matériel réseau employé [8]. Les implémentations des transferts DMA (nombre et tailles de paquets lors du découpage, *pipeline*, ...) varient fortement selon le matériel et en fonction de la taille des transferts pour un même modèle de carte (découpage en paquets, pipeline). Elles pourraient ainsi expliquer ces différentes asymétries.

4.1.3 Répercussion des effets NUIOA sur les communications applicatives

Nous avons observé au niveau de microbenchmarks une majoration de latence de quelques centaines de nanosecondes par lien d'interconnexion traversé, ainsi qu'une dégradation du débit asymétrique et liée aux performances exhibées. L'idée est maintenant d'en estimer les conséquences au niveau applicatif.

Impact sur le débit des communications

Les courbes de débit qui nous ont permis d'introduire la notion d'effets NUIOA au Chapitre 2.3.3 sont rappelées en Figure 4.5. Elles témoignent de l'impact NUIOA lors d'un usage particulier de la bande passante : un ping-pong entre deux hôtes Dalton (Figure 4.1 (a)) utilisant simultanément les deux interfaces réseau disponibles [9]. Cette communication, dite *multirail*, est effectuée sur la

7. Dotées de la dernière version de liens HYPERTRANSPORT, qui assure une bande passante mémoire du même ordre que celle des liens QPI des architectures INTEL.

8. Parmi les différentes plateformes auxquelles nous avons eu accès, le cluster *Mirage* est le seul à disposer de processeurs INTEL Westmere X5650 et de cartes INFINIBAND QDR MT26438. Des tests sur d'autres architectures disposant de cartes similaires, ou sur des plateformes équivalentes pourvues de cartes différentes nous permettraient de confirmer/réprouver cette hypothèse.

9. MYRI-10G et QSNET II.

plateforme de communication NEWMADELEINE [107], qui se charge de répartir les messages de façon transparente sur l'ensemble des réseaux disponibles et en fonction de leurs performances respectives [10]. Il en résulte une augmentation importante de la bande passante, qui permet d'atteindre un débit de 1900 Mo/s. Les échanges sont effectués pour un placement symétrique des tâches et de la mémoire : les processus communicants et la mémoire sont placés pour chaque hôte sur le nœud NUMA auquel sont connectées les interfaces (*placement proche*), ou au contraire sur le nœud éloigné de celles-ci (*placement distant*).

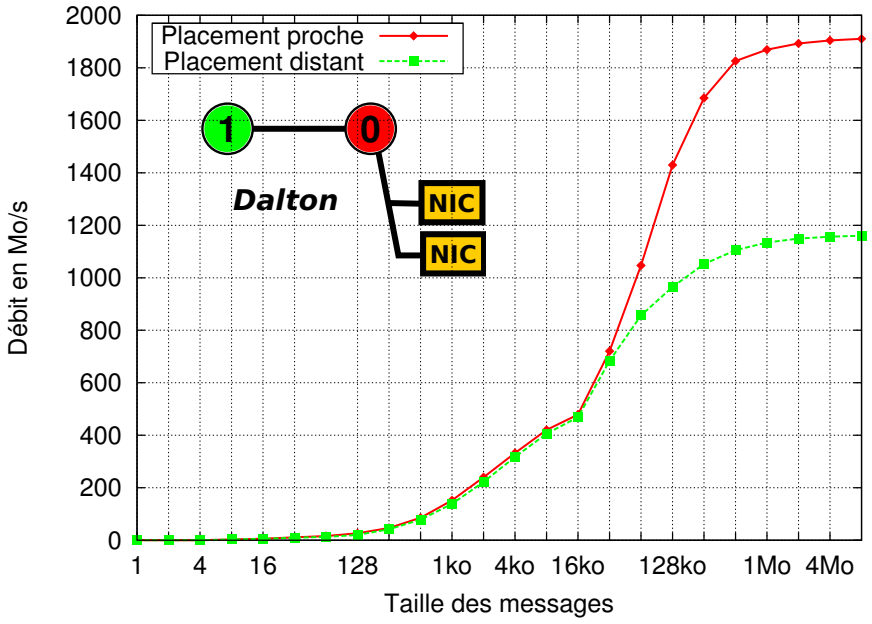
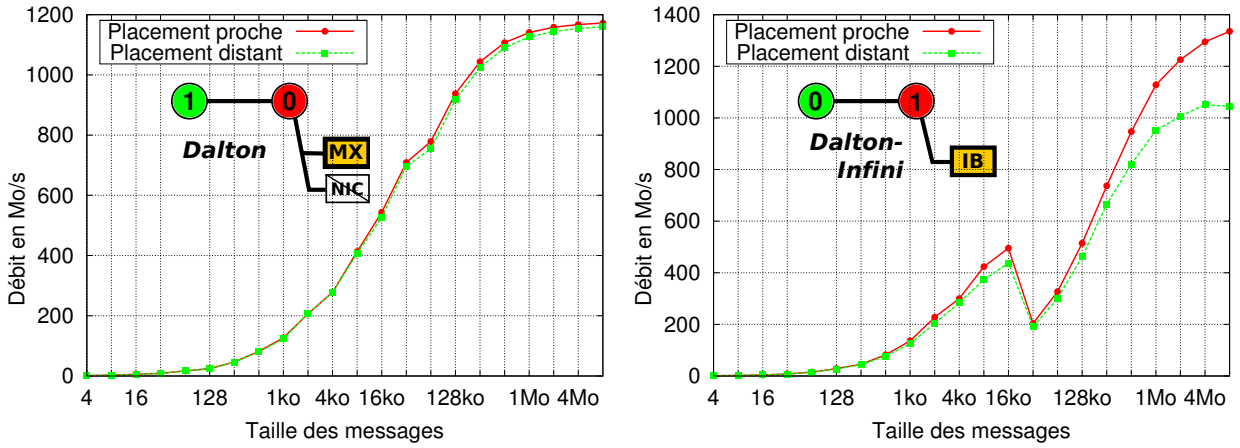


FIGURE 4.5 – *Débits de ping-pong multirail sur la plateforme* NEWMADELEINE *en fonction du placement sur les machines* Dalton.

Le placement distant limite le débit pour des messages de taille supérieure à 32 ko, avec une dégradation pouvant aller jusqu'à plus de 40% par rapport au débit maximum, obtenu avec un placement proche du périphérique réseau (1900 Mo/s pour des messages de 8 Mo). Cette détérioration majeure illustre les proportions que peuvent prendre les effets NUIOA sur les communications applicatives, mais doit être replacée dans son contexte d'usage extrême de la bande passante. En effet, l'observation préalable des transferts locaux nous laisse présager un impact fonction de la bande passante, particulièrement élevée dans cet exemple. De plus, le débit du ping-pong multirail (agrégation des débits des réseaux) relatif à un placement distant de la carte s'avère inférieur à celui d'un ping-pong utilisant la seule carte MYRI-10G (1200 Mo/s). Nous supposons que cela tient d'une congestion liée au fait que les deux contrôleurs d'entrées-sorties sont utilisés simultanément et connectés sur un même lien HYPERTRANSPORT, amplifiant ainsi la dégradation observée.

Les cas plus habituels consistent en des tests de ping-pong *monorails* (envoi sur une carte unique) effectués via la bibliothèque NEWMADELEINE. Les processus communicants sont encore une

10. La quantité de données envoyées sur chaque interface est proportionnelle au débit de chacune.



(a) Débits sur MYRI-10G. *(b) Débits sur* INFINIBAND.

FIGURE 4.6 – *Débits de ping-pong monorails effectués par la bibliothèque de communication* NEWMADELEINE *sur nos machines Dalton et Dalton-Infini, en fonction du placement* NUMA.

fois placés de manière symétrique sur chaque hôte [11]. Nous observons des résultats variés sur les différentes interfaces de notre plateforme multi-OPTERON. Alors que les mesures des transferts locaux sur le réseau MYRI-10G révélaient d'importantes modifications de débit, celles-ci ne sont pas retranscrites au niveau applicatif (Figure 4.6 (a)) pour lequel le débit est moindre (1200 Mo/s contre 1500 Mo/s lors des transferts locaux), et la dégradation est mineure. De plus, nous n'observons aucun changement de débit sur QSNET II qui exhibe un débit très inférieur. Sur le réseau INFINIBAND, qui offre un débit maximum plus élevé, la dégradation reste conséquente, avec une chute supérieure à 20% (Figure 4.6 (b)). **Il semble donc que seuls les débits très élevés (au delà de 1 Go/s) subissent une détérioration relative au placement**.

Pour vérifier que ces observations ne sont pas inhérentes à une particularité de notre génération de machines ou à la bibliothèque de communication NEWMADELEINE, nous avons mené des tests sur différentes architectures OPTERON en utilisant diverses bibliothèques de communication. Ces expériences confirment nos observations. Pour exemple, la Figure 4.7 présente les résultats de ping-pong MPI au dessus de l'implémentation OPEN MPI (version 1.4.1) lancés sur la plateforme *Borderline* (schématisée au côté de la courbe et détaillée en Annexe A.4).

Ils indiquent bien la présence d'effets NUIOA significatifs au delà de taille de messages de quelques dizaines de kilo-octets, et dont l'ampleur est fonction de la technologie réseau sous-jacente. Encore une fois, le débit obtenu sur le réseau MYRI-10G ne varie que très légèrement alors que le débit correspondant sur INFINIBAND affiche une dégradation des performances supérieure à 20% pour un placement des processus sur un nœud éloigné de la carte.

La bande passante brute de INFINIBAND étant plus grande que celle de MYRI-10G, il est possible qu'elle souffre davantage des contentions, mais cette hypothèse semble peu probable pour justifier

11. Dans le cas d'un placement asymétrique des processus (proche sur un hôte et à distance sur le second), les résultats observés pour chacun des tests de ping-pong dénotent d'une dégradation intermédiaire. Ces cas ne sont pas représentés sur les courbes par souci de lisibilité.

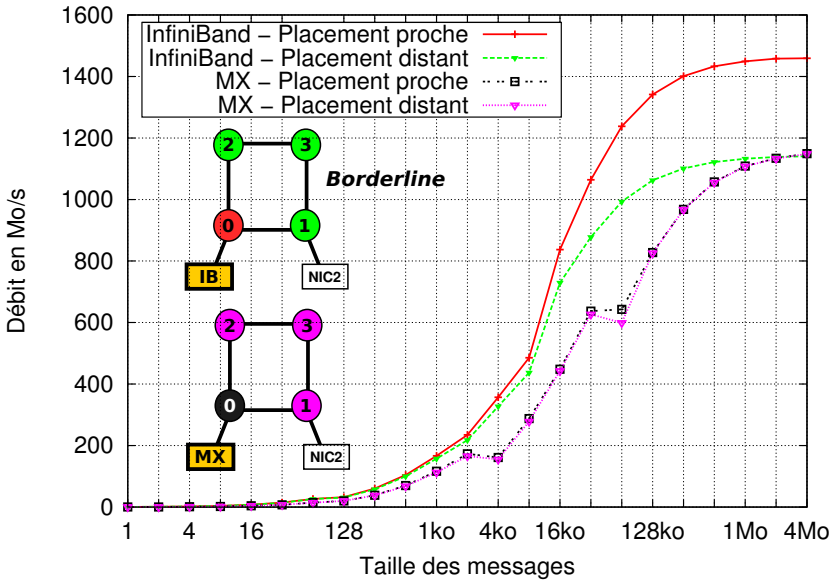


FIGURE 4.7 – *Influence de la localité des processus et des cartes réseau sur le débit d'un ping-pong monorail avec* OPEN MPI *sur la plateforme Borderline.*

seule une telle dégradation. Il semblerait plus probable que les interfaces utilisent des techniques de transferts DMA internes au nœud différentes, à l'origine de problèmes de contentions différents. Par exemple, les transferts DMA d'INFINIBAND pourraient peut-être utiliser de plus petites tailles de paquets. Le nombre de paquets en transit sur les liens d'interconnexion de notre plateforme serait ainsi plus élevé, ce qui pourrait causer une saturation des buffers de requête et de réponse sur les liens.

L'observation des effets NUIOA sur nos plateformes INTEL, plus récentes et dotées de cartes INFINIBAND QDR, présentent les mêmes particularités que nos tests précédents : un placement distant de la carte réseau entraîne une réduction sévère du débit des communications. La Figure 4.8 présente les résultats d'un ping-pong au dessus de l'implémentation OPEN MPI, obtenus sur la machine Hannibal (Figure 4.2 (a)). Ne disposant que d'un seul hôte de ce type, nous avons effectué des échanges entre celui-ci et une machine non NUIOA disposant d'une carte équivalente [12]. La dégradation observée (24%) est ainsi la conséquence de l'éloignement du processus sur un seul hôte [13] et serait largement plus marquée avec un éloignement symétrique sur les deux machines.

Les débits des tests similaires, effectués avec un placement symétrique des processus sur deux machines Mirage (Figure 4.2 (c)), sont dépeints en Figure 4.9. Ils mettent en lumière une réduction de débit de 42% pour un placement distant. Ces résultats tendent ainsi à confirmer que la détérioration NUIOA est d'autant plus forte que le débit est élevé.

12. L'architecture utilisée est un bi-processeur hexa-cœurs Westmere X5650 (2 nœuds NUMA) pour laquelle les processeurs sont tous deux directement reliés au contrôleur d'entrées-sorties au travers d'un lien, comme nous le verrons en Section 4.1.4. Le placement sur les nœuds est donc indifférent (par acquis de conscience, les placements sur chaque nœud ont été testés et présentent bien des résultats identiques).

13. A la différence des expériences de ping-pong présentées pour les architectures OPTERON pour lesquelles le placement distant des processus est proposé symétrique sur les hôtes.

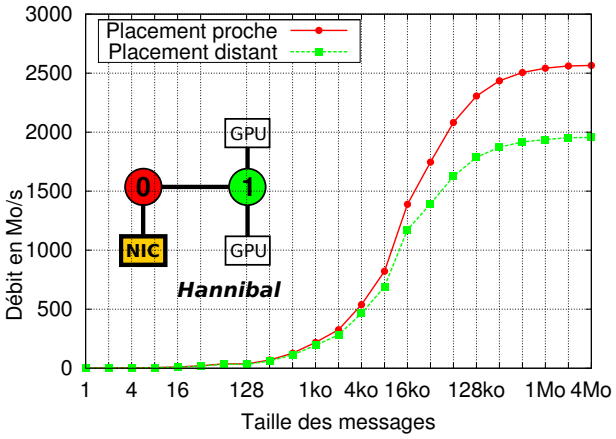


FIGURE 4.8 – *Débits d'un ping-pong* OPEN MPI *sur le réseau* INFINIBAND *en fonction du placement* NUIOA *sur la machine Hannibal (placement proche sur l'hôte partenaire).*

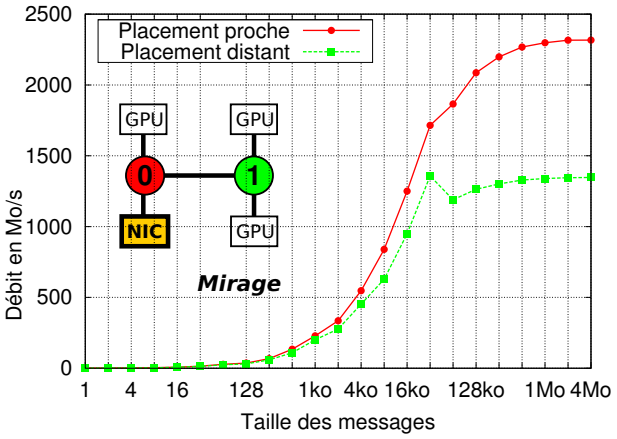


FIGURE 4.9 – *Débits d'un ping-pong* OPEN MPI *sur le réseau* INFINIBAND *entre deux hôtes du cluster Mirage et pour un placement* NUIOA *symétrique, local ou distant.*

Effets asymétriques

Pour enrichir nos expérimentations nous avons développé des tests de RDMA [14] (*Remote Direct Memory Access*) sur le réseau INFINIBAND, effectués sur nos machines à deux nœuds *Dalton-Infini* (Figure 4.1 (b)) dont les résultats sont présentés en Figure 4.10. Pour ces tests unidirectionnels, nous faisons varier la position physique de l'émetteur (E) et du récepteur (R) sur nos hôtes. Les mesures de débit montrent des effets similaires à ceux décrits précédemment, avec une saturation avant 1100 Mo/s et une perte de 25% par rapport au débit maximum. Nous observons sur ces architectures que le placement de la mémoire cible de l'écriture RDMA importe, contrairement à celui du côté émetteur. Nous avons constaté des réactions semblables dans le cas d'une lecture RDMA. Il apparaît donc que seul l'emplacement du tampon mémoire dans lequel les données sont *déposées* compte. Cet effet pourrait être provoqué par une saturation du bus HYPERTRANSPORT. Ces spécifications [39] indiquent en effet que les nombres de tampons matériels de requêtes et de réponses peuvent différer. Les demandes de lectures et écritures envoyées au travers des liens mémoire HYPERTRANSPORT utilisent des paquets requêtes et réponses qui subissent des limites matérielles différentes à la source, en transit, et en destination (nombre limité de paquets réponses en vol simultanément par exemple).

Ces effets applicatifs asymétriques ne sont pas restreints à nos transferts RDMA et s'appliquent à d'autres tests. Pour en témoigner, la Figure 4.11 présente les débits du test *Stream* qui effectue un ping-pong entre deux hôtes pour lequel la réponse (*pong*) est un transfert minime (envoi d'un message vide) sur nos machines *Dalton*. Les résultats obtenus correspondent ainsi à un aller-retour entre les machines pour lesquels le temps de transfert du message *retour* est d'autant plus négligeable que le message *aller* est important. Pour de larges messages, le temps d'un ping-pong divisé par la taille du message envoyé est ainsi assimilable au débit d'un transfert unidirectionnel.

14. Ces tests sont semblables à ceux proposés au niveau des transferts locaux sur INFINIBAND. Ils sont ici repris dans un contexte plus général faisant varier les positions côtés émetteur et récepteur.

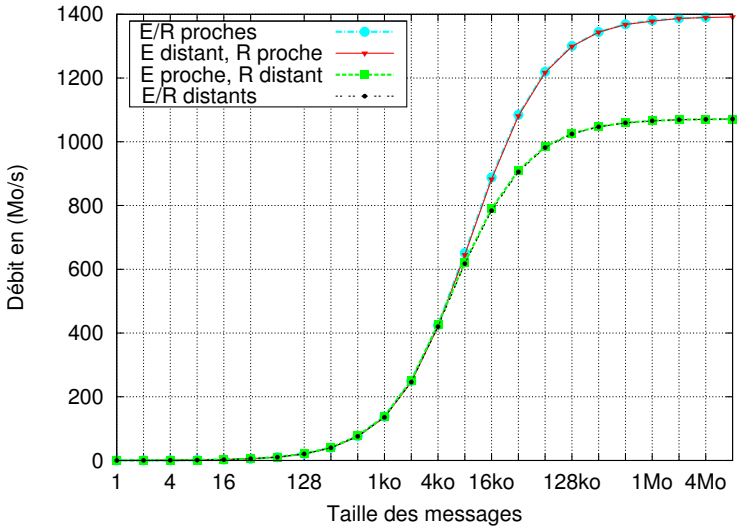


FIGURE 4.10 – *Variation du débit des transferts RDMA sur le réseau* INFINIBAND *en fonction des placements* NUMA *côté émetteur (E) et récepteur (R) sur des machines à deux nœuds.*

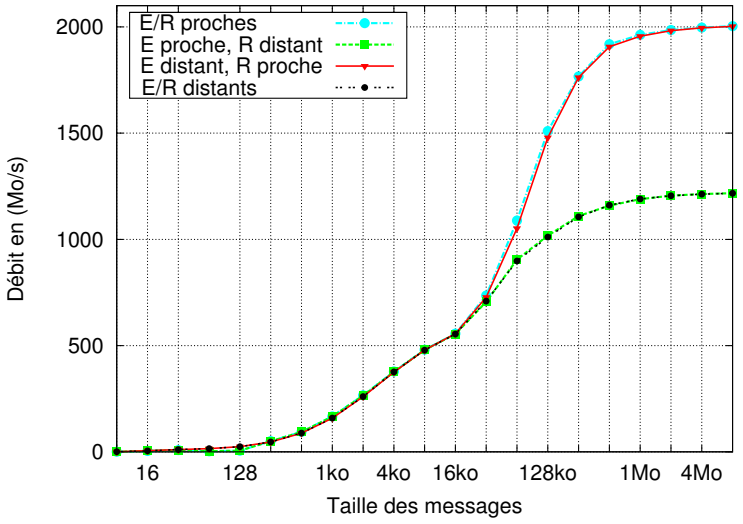


FIGURE 4.11 – *Performances des transferts du test Stream en fonction du placement du processus effectuant le transfert des données (émetteur - E) et du processus partenaire (récepteur - R), envoyant des messages vides.*

Sur ces machines à deux nœuds *Dalton*, ce test effectué en multirail sur les réseaux QSNET II et MYRI-10G présente bien une dégradation relative au seul placement des zones mémoire destinations des données [15] de l'ordre de 40%. Il souligne ainsi la répercussion au niveau applicatif des effets asymétriques présentés en Section 4.1.2, invisibles dans les tests de ping-pong bidirectionnels précédents.

15. Adjoint au placement de récepteur.

Comme annoncé par les résultats présentés en Section 4.1.2, les tests de RDMA effectués sur nos architectures INTEL *Mirage* présentent un impact NUIOA majoritaire sur les lectures DMA (effectuées depuis la émoire de l'hôte vers le périphérique) et moindre sur les écritures. Ils présentent ainsi le schéma inverse de celui des Dalton, avec un placement de l'émetteur (E) et donc de la zone mémoire *source* déterminant. Bien que le placement de l'émetteur soit décisif, celui du récepteur (R) joue lui aussi un rôle sur les performances. Dans le cas où seul le récepteur est distant (courbe verte), le débit est également réduit (de 15%) par rapport au placement idéal (E/R proches), sans toutefois souffrir d'une dégradation aussi importante que celle observée lorsque l'émetteur est distant (40%). Les tests de lectures RDMA témoignent des mêmes caractéristiques, avec un placement de la zone mémoire source impactant grandement les performances et un effet moindre mais notable de la localité de la zone mémoire cible.

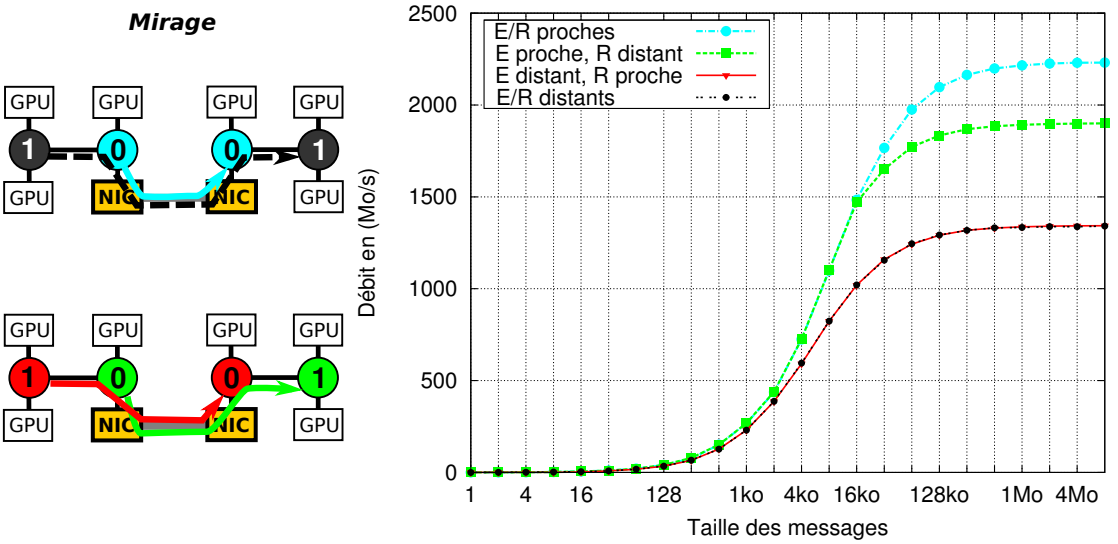


FIGURE 4.12 – *Débits de transferts RDMA (écriture) en fonction du placement* NUIOA *des processus sur deux hôtes du cluster Mirage.*

La Figure 4.13 propose les débits d'écritures RDMA [16] sur notre hôte *Hannibal* [17]. Nous distinguons ici les écritures ciblant la machine *Hannibal* (courbes rouge et verte) ou au contraire initiées depuis cet hôte (noir et magenta). Il apparaît tout d'abord une sévère différence de débit entre les écritures sur notre bi hexa-cœur Westmere (qui atteignent 2800 Mo/s et 2900 Mo/s) et les écritures ciblant la mémoire d'Hannibal dont les débits sont moindres (1525 Mo/s et 2340 Mo/s). Dans ce second cas, une saturation (particulièrement visible sur la courbe verte) limite la bande passante. Il semblerait que celle-ci soit causée par un bug matériel [18] inhérent à la présence de deux contrôleurs d'entrées-sorties sur l'architecture. En faisant varier le placement de la *zone cible* sur Hannibal (écriture sur H), nous observons comme attendu une dégradation majeure du débit (ici de 22%). Une dégradation moindre (de 3,5%) est également visible pour un placement distant de la *zone source* (écriture depuis H). Il apparaît donc que les effets NUIOA ne sont pas limités au

16. Les observations sur des lectures RDMA concordent avec les caractéristiques présentées pour les écritures.
17. Comme pour les tests de ping-pong, un bi-processeur Westmere X5650 est utilisé comme machine partenaire.
18. Défaut observé sur différentes plateformes dotées de deux contrôleurs d'entrées-sorties, et d'autres types de périphériques (GPU par exemple, `http://forums.nvidia.com/index.php?showtopic=104243`)

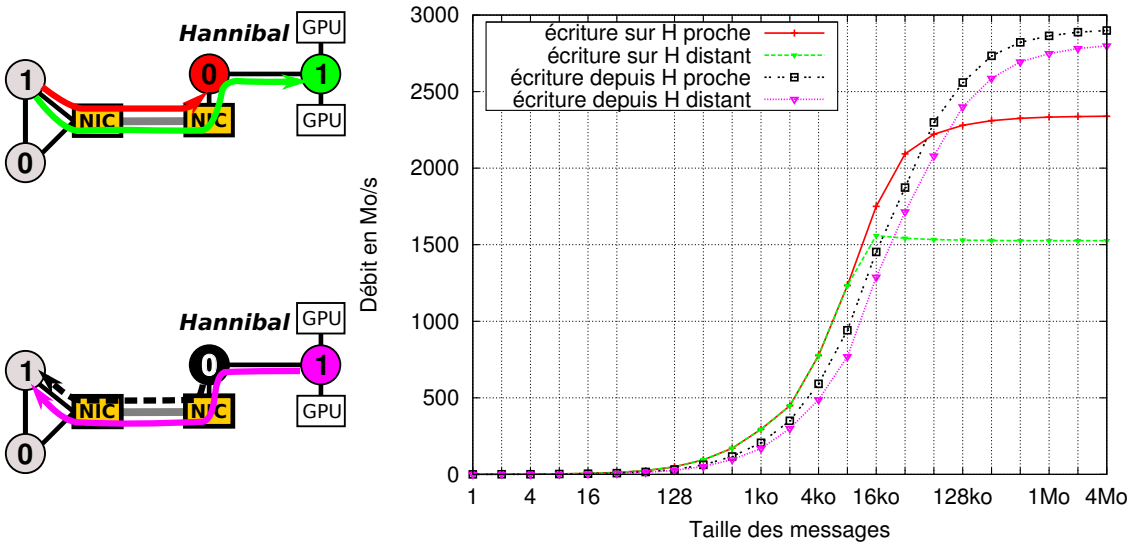


FIGURE 4.13 – *Débits de transferts RDMA (écriture) en fonction du placement* NUIOA *des processus sur notre machine Hannibal (H).*

placement de la zone destination mais impactent ici majoritairement celui-ci.

Détérioration hiérarchique ?

Étendus à notre machine à huit nœuds *Hagrid* (Figure 4.1 (c)), les tests de RDMA indiquent que le débit d'une écriture ne décroît pas avec une augmentation du nombre de liens traversés (Figure 4.14). **Si le placement est déjà distant, augmenter la distance à l'interface réseau ne réduit pas davantage les performances**. Les tests équivalents sur les machines à quatre nœuds *Borderline* (Figure A.4) confirment ce résultat. Les courbes de débit "placement distant" présentées Figure 4.7 sont ainsi indifféremment obtenues avec un placement sur n'importe lequel des trois nœuds éloignés de l'interface utilisée. Ce résultat est cependant valable dans le seul cas où la machine n'est pas chargée. Dans le cas contraire les contentions sur les liens HYPERTRANSPORT entraînent une baisse conséquente des performances avec la distance.

Nous avons reproduit notre test de RDMA sur INFINIBAND, tout en générant des écritures concurrentes en mémoire pour créer des contentions sur certains liens HYPERTRANSPORT, de sorte à simuler les effets d'une charge de travail sur la machine. La Figure 4.15 présente un exemple de perturbation étudiée. Une écriture RDMA cible ici la mémoire du nœud 6 de l'hôte Hagrid. Les résultats observés sous différents types de perturbations sont résumés par la Table 4.4. La contention entraîne une chute du débit, tant sur les écritures en mémoire que sur le débit des communications, qu'il s'agisse d'écritures en mémoire locale ou distante, dans la même direction sur le bus HYPERTRANSPORT ou non.

En conditions réelles, c'est-à-dire lorsqu'une application charge la machine et utilise donc intensément les liens mémoire, un placement des tâches de communication éloigné du périphérique réseau tend donc bien à réduire les performances de communication davantage qu'un placement plus proche du périphérique. La notion de placement "hiérarchique" n'est donc pas à exclure. Ces

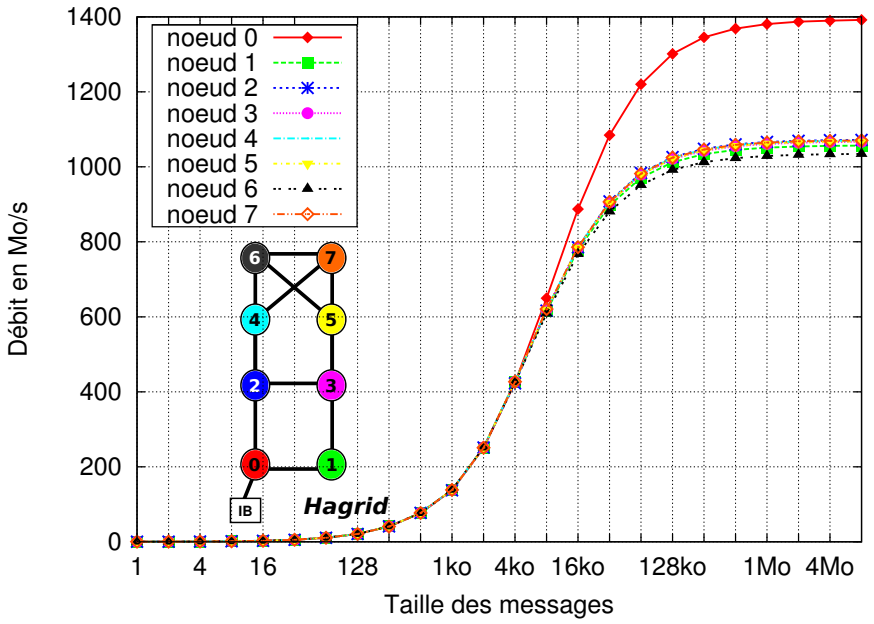


FIGURE 4.14 – *Variation du débit d'un transferts RDMA sur le réseau* INFINIBAND *en fonction du placement côté récepteur sur notre machine à huit nœuds.*

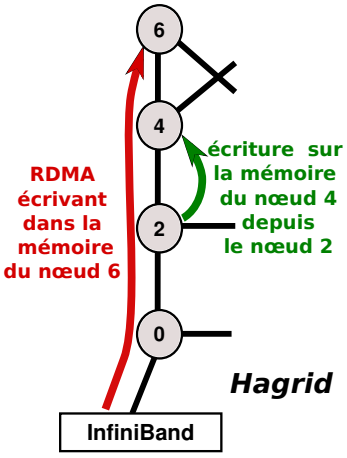


FIGURE 4.15 – *Perturbation d'un RDMA (écrivant depuis le réseau sur la mémoire du nœud 6) par une écriture du processeur du nœud 2 sur la mémoire du nœud 4.*

		Débit écriture	Débit RDMA
écriture RDMA en mémoire nœud 6, seul			**1049**
processeur nœud 2 écrit en mémoire nœud 4	seul	**2171**	
	pendant RDMA	1232	883
processeur nœud 4 écrit en mémoire nœud 2	seul	**2177**	
	pendant RDMA	1611	971
processeur nœud 2 écrit sur le nœud local (2)	seul	**2740**	
	pendant RDMA	2040	1002
processeur nœud 4 écrit sur le nœud local (4)	seul	**2730**	
	pendant RDMA	1616	859

TABLE 4.4 – *Impact d'écritures en mémoire par un processeur sur le débit de transferts RDMA entrant sur la machine* Hagrid *(débits en Mo/s).*
Le processeur d'un nœud NUMA *effectue une écriture dans la mémoire locale ou celle d'un autre nœud pendant que le RDMA écrit depuis le réseau en mémoire 6.*

résultats sont toutefois difficiles à prédire en détails comme en témoignent ici les dégradations inégales pour les accès locaux sur deux nœuds différents (2 et 4). D'un point de vue général, anticiper les contentions liées à la charge d'une machine constitue en soit un problème délicat qui

rend la prévision des effets particulièrement épineuse.

Impact sur la latence des communications
Au niveau applicatif, l'examen des latences des communications sur le réseau MYRI-10G révèle une influence négligeable du placement NUMA, avec un surcoût d'environ 25 ns pour une latence théorique de 2,5μs, soit une fluctuation de 2% par aller-retour sur nos hôtes *Dalton*. Sur ces machines, le surcoût est d'autant plus négligeable sur INFINIBAND puisque la latence est plus élevée. Le réseau QSNET II montre une influence NUIOA plus grande puisque sa latence théorique est très inférieure, entre 1 et 2μs selon le mode de communication. Avec l'utilisation des opérations *Put/Get* natives, la latence brute est très proche de 1μs et l'impact d'un placement distant atteint 100 ns de chaque côté entraînant une différence globale de 20%.

Sur nos architectures INTEL dotées de cartes INFINIBAND récentes qui exhibent des latences relativement faibles, nous observons un surcoût de l'ordre de 150 ns pour un lien traversé. La répercussion provoque ainsi une augmentation de latence de 11% pour un placement distant sur l'hôte *Hannibal* seul, et atteignant 15% pour un placement symétrique distant sur les machines de la grappe *Mirage*.

Si la dégradation de latence peut être marquée pour des applications extrêmement sensibles à la latence, la perte occasionnée par un placement distant est du même ordre que celle d'un défaut de cache et peut être considérée négligeable dans la plupart des cas.

4.1.4 Spectre d'impact NUIOA

Architectures NUMA et NUIOA
Notre étude des effets NUIOA a débuté sur des architectures multi-OPTERON, dont la liaison point-à-point des éléments au travers des liens HYPERTRANSPORT implique la connexion directe de chaque banc mémoire et contrôleur d'entrées-sorties sur une puce [41]. Il en résulte des plateformes NUMA et NUIOA lorsque les cœurs de la puce ont un accès privilégié à la mémoire et aux périphériques connectés sur celle-ci.

Si la plateforme de test sur laquelle nous avons travaillé est relativement ancienne [19], les architectures AMD plus récentes présentent une connectivité identique. Bien qu'elles puissent supporter jusqu'à quatre liens HYPERTRANSPORT, les contrôleurs d'entrées-sorties restent liés à une unique puce. Les Figures 4.16 (a) et (b) schématisent de telles plateformes.

De la même façon que la technologie HYPERTRANSPORT l'a fait pour les architectures AMD, l'introduction de la technologie QUICKPATH INTERCONNECT [44] (QPI) a généralisé les architectures NUMA des serveurs du constructeur INTEL. En fonction de la manière dont sont attachés les contrôleurs d'entrées-sorties sur les liens QPI, ces plateformes peuvent également présenter des effets NUIOA, mais cette caractéristique n'est pas générale à l'ensemble des serveurs. En effet, le plus souvent deux processeurs sont interconnectés et tous deux reliés au contrôleur d'entrées-sorties au travers d'un lien, comme dépeint en Figure 4.16 (c), donnant lieu à une structure NUMA

19. Processeurs AMD OPTERON 265 et 865 interconnectés par des liens HYPERTRANSPORT a 1000 MHz (génération 1.0)

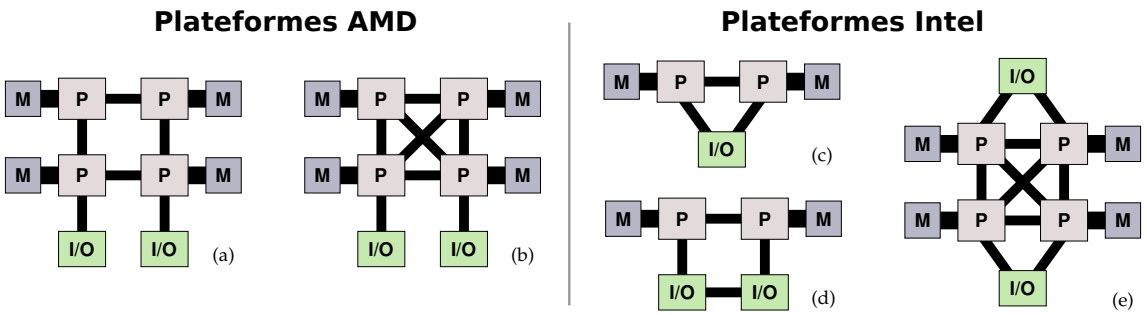


FIGURE 4.16 – *Interconnexion de processeurs (P), bancs mémoire (M) et contrôleurs d'entrées-sorties (I/O) sur différentes plateformes :*
(a) 4 processeurs AMD Istanbul
(b) 4 processeurs AMD Magny-Cours
(c) et (d) 2 processeurs INTEL *Westmere-EP*
(e) 4 processeurs INTEL *Nehalem-EX ou Westmere-EX.*

et non NUIOA. Les serveurs plus larges disposent cependant d'un plus grand nombre de puces et/ou slots d'entrées-sorties créant des plateformes NUIOA comme l'illustrent les Figures 4.16 (d) et (e).

Avec la tendance à l'intégration des composants au sein des puces, la nouvelle génération de processeurs devrait généraliser les architectures NUIOA. En effet, les processeurs INTEL *Sandy-Bridge* [20] sont par exemple conçus avec un contrôleur d'entrées-sorties intégré à la puce (Figure 4.17), donnant ainsi un accès privilégié systématique aux cœurs locaux.

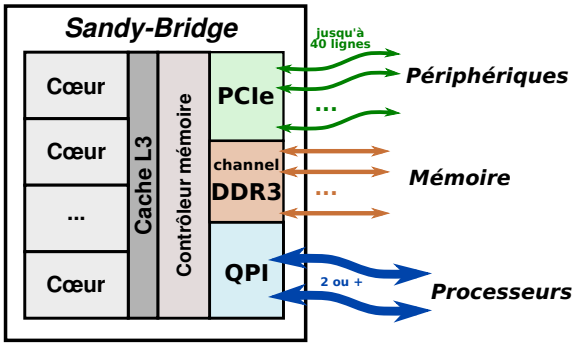


FIGURE 4.17 – *Structure des processeurs Sandy-Bridge attendue dans les futurs serveurs* INTEL. *Contrairement aux générations précédentes, le contrôleur d'entrées-sorties n'est pas "attaché" aux processeurs par un ou plusieurs liens mais directement intégré sur la puce. Chaque périphérique est ainsi directement relié à un unique processeur, ce qui rend l'architecture immédiatement* NUIOA.

Quels effets sur d'autres périphériques ?
Comme nous l'avons vu, les effets NUIOA pénalisent les performances des communications sur

20. Seules les versions pour ordinateurs portables sont à ce jour commercialisées.

réseau rapide et sont ainsi parfois désignés sous le terme NUNA (*Non Uniform Network Access*). En pratique, ces propriétés ne sont pas limitées aux interfaces réseau, mais sont susceptibles de toucher les différents périphériques connectés sur un nœud NUMA et à distance des autres.

Notre machine Hannibal disposant de multiples GPU NVIDIA attachés sur les différents nœuds NUMA (Figure 4.2 (a)), nous avons pu étudier l'impact du placement sur ces périphériques. Les résultats représentés Figure 4.18 sont obtenus en mesurant le débit de transferts DMA effectués depuis un GPU vers la mémoire de l'hôte (*écriture*), ou au contraire depuis la mémoire vers le périphérique (*lecture*) à l'aide du test NVIDIA *BandwithTest*. Ces expériences témoignent de

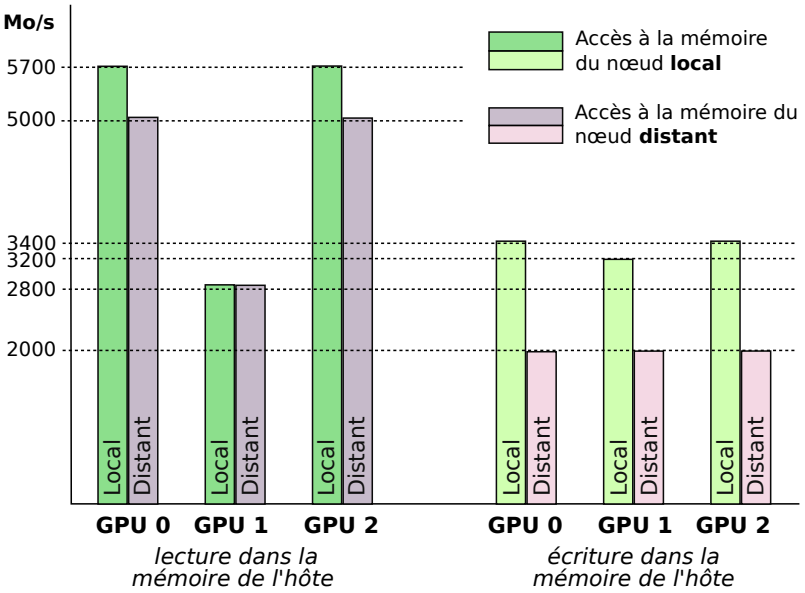


FIGURE 4.18 – *Impact* NUIOA *observé sur le débit de large transferts de données (> 4 Mo) entre mémoire et GPU sur la machine Hannibal à l'aide du test NVIDIA BandwithTest.*

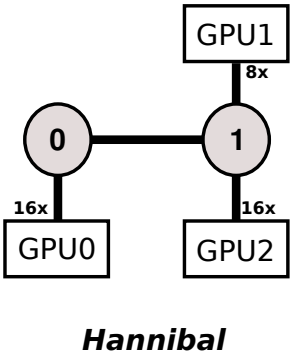


FIGURE 4.19 – *Bi quadri-cœur Nehalem doté de 3 GPUs NVIDIA.*

l'aspect hétérogène des effets NUIOA sur diverses technologies d'entrées-sorties avec un impact NUIOA sur le débit pouvant atteindre 42% dans le cas des transferts du GPU vers la mémoire. Elles corroborent nos observations précédentes avec des dégradations asymétriques et proportionnelles au débit (ici particulièrement élevé). Les transferts vers et depuis le GPU 1 exhibent une bande passante moindre (due à l'utilisation d'un lien PCIE de largeur 8x, contre 16x pour les GPUs 0 et 2) et de ce fait une dégradation moins importante pour un placement NUMA distant.

Les résultats des tests équivalents lancés sur une machine *Mirage* sont illustrés Figure 4.20. Les GPUs, identiques et attachés à des connecteurs PCIE analogues, offrent des performances équivalentes. Comme pour nos tests précédents, nous remarquons ici que les écritures DMA sont plus affectées par le placement distant (dégradation de 30%) que les lectures (15%). Cette caractéristique, contraire à celle observée sur les cartes réseau de cette machine [21], tend ainsi à confirmer notre hypothèse selon laquelle les effets asymétriques sur le débit des transferts seraient liés au périphérique employé plutôt qu'à un effet de l'architecture.

21. Effets NUIOA affectant d'avantage les lectures DMA que les écritures sur la carte réseau INFINIBAND (voir Table 4.3) sur ces machines.

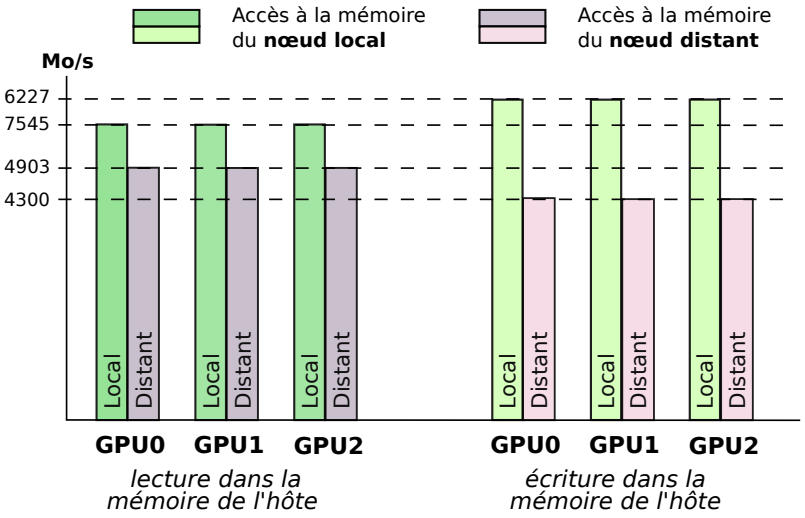


FIGURE 4.20 – *Impact* NUIOA *observé sur le débit de large transferts de données (> 4 Mo) entre mémoire et GPU sur une machine Mirage à l'aide du test NVIDIA BandwithTest.*

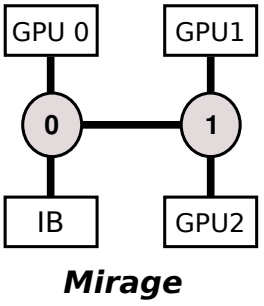


FIGURE 4.21 – *Bi hexa-cœur Westmere doté de 3 GPUs NVIDIA.*

Nous avons également réalisé des tests sur les accès aux disques de la machine *Bertha* [22] connectés sur différents nœuds (Figure A.12), à l'aide de la suite de tests de système de fichiers IOzone [8]. Les débits relatifs aux différents types d'accès disque souffrent d'une forte saturation et n'excèdent pas 600 Mo/s sur notre plateforme. Ils ne nous permettent ainsi pas d'observer d'effets NUIOA, uniquement visibles pour de très larges bandes passantes. Nous avons cependant remarqué une dégradation de 6 à 10% du débit des écritures dans les tampons locaux du système d'exploitation. Ce résultat est surprenant puisque ces tampons sont normalement alloués sur le nœud NUMA local à celui du processus. Une hypothèse à ce sujet est que le système de fichier pourrait nécessiter des métadonnées qu'il doit préalablement récupérer sur le disque, créant des accès distants additionnels. Si nous n'avons pu voir d'impact NUIOA concret lors des copies disques sur notre architecture nous pensons qu'il devrait cependant apparaître sur du matériel plus performant offrant une large bande passante.

La technologie *Input/Output Acceleration Technology* [147] (I/OAT), apportée aux contrôleurs mémoire INTEL depuis quelques années, comporte un ensemble de spécificités conçues pour améliorer les performances réseau et réduire la consommation processeur [146, 148]. Parmi celles-ci, un moteur DMA (*DMA Engine*) intégré au contrôleur mémoire permet d'effectuer en arrière plan des copies efficaces sans intervention du processeur. Alors que les transferts sont initiés par le moteur DMA, la position physique du contrôleur auquel il est intégré par rapport à celle de la mémoire peut jouer sur les performances de transfert. Pour en témoigner, nous avons effectué des séries de transferts DMA [23] sur notre plateforme Hannibal, initiés par les différents moteurs DMA disponibles et en fonction du placement des processus communicants et des données associées. La Figure 4.23 présente les courbes de débit obtenues pour les différents placements. On observe ici une perte de 35% du débit lorsque le moteur DMA à l'origine du transfert est éloigné des 2 pro-

22. La configuration de Bertha et une sortie graphique de lstopo sont disponibles en Annexe A.9
23. A l'aide du module noyau KNEM (`http://runtime.bordeaux.inria.fr/knem/`).

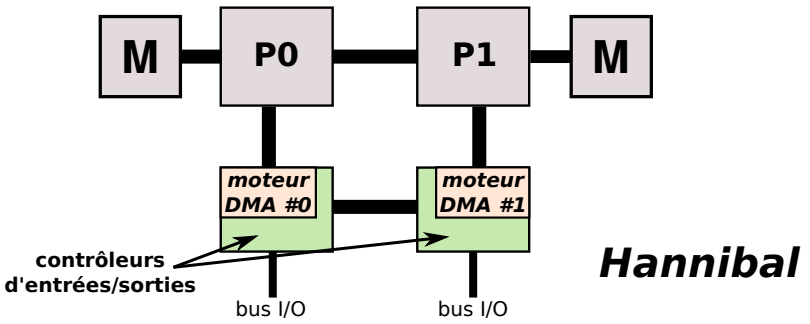


FIGURE 4.22 – *Architecture Hannibal dotée de deux contrôleurs d'entrées-sorties disposant de la technologie* I/OAT. *Le transfert de données est initié par le moteur DMA de l'un des deux contrôleurs et effectué en tâche de fond sans intervention du processeur.*

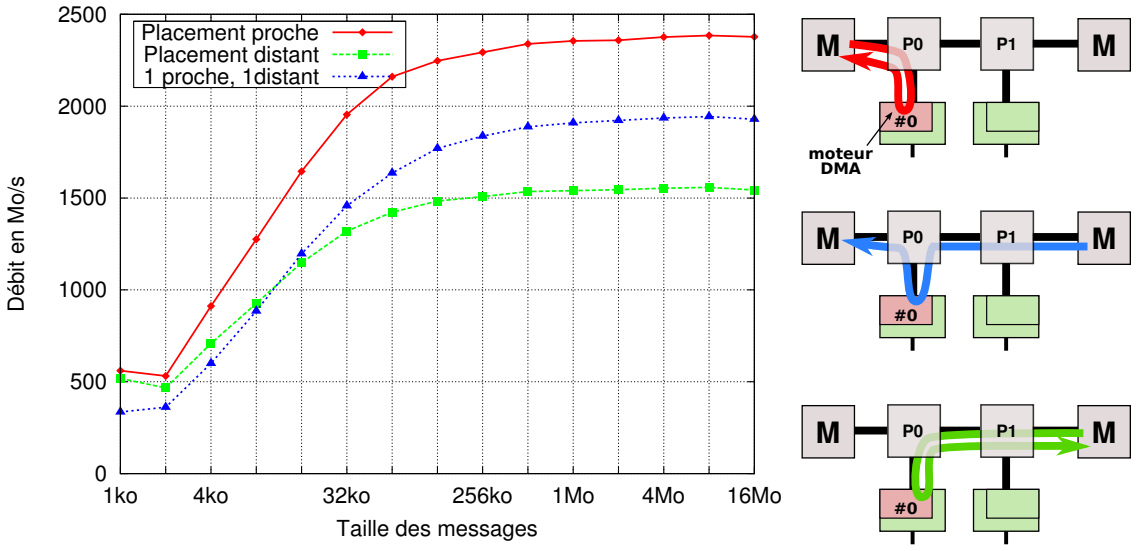


FIGURE 4.23 – *Débits des transferts de données en fonction du placement des paires de processus communicants et relatif à la localité du moteur DMA sollicité (#0)*

cessus (en comparaison avec un placement des processus proche du moteur DMA) et un résultat intermédiaire lorsqu'un seul des processus est distant, avec une perte de débit de 19%.

4.1.5 Bilan

L'utilisation des technologies d'interconnexion HYPERTRANSPORT et QPI dans les architectures multiprocesseurs récentes a réintroduit les architectures NUMA dans le domaine du calcul haute performance. Ils conduisent à la création de structures NUMA et NUIOA (*Non Uniform Input/Output Access*) lorsque les contrôleurs d'entrées-sorties sont attachés sur une puce au travers d'un lien d'interconnexion, et donc proches des cœurs de cette puce et à distance des autres. Nous avons étudié l'impact de cette distance sur les performances des communications réseau et observé les particularités suivantes :

- La latence des communications souffre d'un très léger surcoût de l'ordre de quelques dizaines de nanosecondes supplémentaires par lien d'interconnexion traversé. Si celui-ci peut être nocif pour des applications très sensibles à la latence, il peut cependant être considéré comme négligeable dans la plupart des cas.
- Un placement distant des interfaces réseau peut entraîner une réduction du débit potentiellement très importante. Celle-ci est d'autant plus visible pour de larges transferts que la bande passante maximum soutenue est élevée.
- L'augmentation de la distance n'affecte pas davantage la dégradation du débit en l'absence de contention. Dans le cas contraire, la concurrence sur les liens mémoire dégrade cependant les performances.
- Les effets NUIOA sur le débit apparaissent également asymétriques. C'est en réalité le placement des zones mémoires invoquées lors des transferts qui est déterminant, plus que celui des tâches de communication. Selon le matériel employé, la dégradation de débit est plus marquée pour une position distante de la mémoire *cible* ou au contraire *source* des transferts.
- Enfin, la présence d'effets NUIOA a été confirmée sur différentes architectures mais également pour différents types de périphériques. Ils sont notamment visibles sur les échanges entre la mémoire et les GPUs, et apparaissent aussi pour des transferts effectués via la technologie I/OAT.

Dans le cadre du calcul haute performance nécessitant des transferts de données aussi rapide que possible, l'impact NUIOA sur le débit des transferts (qui peut être réduit jusqu'à plus de 40% sur des communications multirails), peut constituer une sévère entrave aux performances applicatives. Pour obtenir des résultats optimaux, le placement des tâches communicantes et de la mémoire associée relatif à la position des périphériques ne devrait ainsi pas être négligé.

4.2 Implémentation d'une solution de placement automatique

La prise en charge des effets NUIOA est indispensable à l'obtention de communications optimales. Les processus utilisés lors des tests réseau sont ainsi généralement placés manuellement pour en tirer les meilleures performances, mais aussi assurer leur reproductibilité. En effet, sans liaison des tâches sur les processeurs (*binding*), celles-ci peuvent subir des migrations aléatoires initiées par l'ordonnanceur, par exemple lors du réveil d'un processus démon. À l'opposé, un placement manuel est contraignant car l'utilisateur doit prendre connaissance de l'architecture matérielle et de l'emplacement des périphériques. De plus, un tel placement ne peut convenir aux cas complexes des applications multithreadées effectuant des communications irrégulières, pour lesquelles un compromis entre le placement proche des périphériques pour les communications et l'exploitation de l'ensemble des processeurs des différents nœuds NUMA est nécessaire. La portabilité des performances devrait être garantie par le système d'exploitation ou les intergiciels grâce à des mécanismes de placement des tâches et des tampons mémoire. Pour répondre à ces contraintes, nous proposons une solution de placement automatique que nous avons initialement développée dans la bibliothèque de communication NEWMADELEINE [108].

4.2.1 Trouver le nœud le plus proche de chaque carte réseau

Effectuer un placement proche d'une interface nécessite de savoir quel nœud NUMA en est le plus près physiquement. Les tests de transferts locaux présentés dans la Section 4.1.2, et tout particulièrement ceux de latence nous permettraient de détecter l'emplacement physique de la carte, mais les contraintes qu'ils imposent (machine non chargée et droits privilégiés) les rendent inadaptés à un usage courant. L'idéal serait d'exposer la topologie NUMA aux supports exécutifs alors chargés du placement.

Les BIOS [24] sont censés pouvoir détecter quels nœuds NUMA sont proches de chaque bus d'entrées-sorties et les transmettre aux systèmes d'exploitation. Il est ainsi possible de déterminer l'affinité de chaque périphérique PCI. La difficulté est alors de pouvoir traduire cette information en un renseignement exploitable au niveau applicatif. En effet, les processus ne manipulent pas de périphériques PCI mais des descripteurs virtuels de haut niveau (socket, endpoint, file descriptor), pointant sur des canaux de communication physiques cachés par le système. La correspondance entre ces descripteurs et les périphériques physiques ne peut être faite que par les pilotes réseau. Certains pilotes bas-niveau tels que l'interface VERBS d'INFINIBAND, fournissent cette équivalence [49] grâce aux fichiers spéciaux `sysfs`, donnant ainsi la possibilité de retrouver la localité des descripteurs lorsque la localité PCI est connue. Les pilotes de QSNET II et ETHERNET ne permettent pas une telle correspondance, nous privant actuellement de moyen d'obtenir automatiquement l'emplacement de la carte réseau du système. Pour le réseau MYRI-10G, nous avons ajouté un support explicite dans le pilote MX [54] de MYRICOM fournissant l'attribut NUIOA associé à un canal de communication ouvert (*endpoint MX*) à l'aide de commandes dédiées.

Nous avons intégré un patch dans le noyau LINUX 2.6.22 pour exposer le nœud NUMA proche des interfaces, de sorte que les intergiciels puissent utiliser facilement les outils `libnuma` pour placer les tâches. Cette détection matérielle a été implémentée dans la bibliothèque NEWMADELEINE développée dans notre équipe, pour permettre un placement automatique des tâches proches des cartes.

Cette stratégie a depuis été généralisée dans la bibliothèque `hwloc` [33] qui détecte de nombreuses caractéristiques matérielles et fournit les outils nécessaires à leur exploitation. Celle-ci permet ainsi de retourner directement un ensemble de cœurs proches d'une interface dont la correspondance entre les descripteurs virtuels et physiques est possible, ou disposant de commandes dédiées retournant la localité NUMA (MX, CUDA). Elle offre également de prendre en charge le placement des tâches sur ces cœurs sans avoir à effectuer d'appel explicite à un autre outil de placement, et est aujourd'hui responsable de ces interventions au sein de NEWMADELEINE.

4.2.2 Mise en œuvre dans NEWMADELEINE

Une fois que la bibliothèque NEWMADELEINE connaît la position physique des cartes réseau, elle doit placer les tâches et la mémoire correctement. Il est important d'effectuer ce placement au bon moment. Trop hâtif, il risquerait de forcer le placement de ressources qui n'ont pas lieu

24. Le BIOS (*Basic Input/Output System*) propose un ensemble de fonctions permettant d'effectuer des opérations élémentaires lors de la mise sous tension du système, telles que la vérification et le chargement des périphériques disponibles.

de l'être, par exemple des threads ou des zones mémoire non impliqués dans les communications. Un placement tardif pourrait quant à lui imposer une migration délicate de ressources complexes telles que des pages verrouillées en mémoire physique en vue d'un transfert DMA. La bibliothèque doit donc laisser l'application démarrer, obtenir l'attribut NUIOA de toutes les cartes, effectuer le placement en conséquence, et enfin initialiser les pilotes (*drivers*). Cette initialisation entraîne alors un placement correct et définitif des ressources spécifiques aux pilotes, en particulier les tampons mémoire utilisés par la suite lors de chaque transfert DMA. L'intergiciel doit également être capable d'exposer ces informations de placement à l'application pour que celle-ci puisse allouer correctement ses propres tampons.

Nous avons implémenté cette stratégie dans NEWMADELEINE et vérifié que nous obtenons toujours les performances optimales (équivalentes à celles d'un placement manuel). Chaque application utilisant les réseaux MYRI-10G ou INFINIBAND est automatiquement placée sur le nœud NUMA le plus proche du périphérique correspondant. Lorsqu'une technologie de réseau qui ne fournit pas les attributs NUIOA est utilisée, NEWMADELEINE part du principe qu'aucun nœud n'est plus proche de l'interface réseau que les autres et ne prend donc pas cette interface en compte dans sa prise de décision de placement.

4.2.3 Cas des communications multirails

Les communications multirails dont les débits attendus sont élevés sont fortement sujettes aux aspects NUIOA (Section 4.1.3). Leur placement est d'autant plus complexe que les interfaces réseau utilisées n'exposent pas forcément les mêmes affinités NUIOA. Il est commun que les machines disposent de plusieurs bus d'entrées-sorties connectés à des nœuds NUMA distincts comme c'est le cas pour les machines *Borderline* (Figure A.6). Pour gérer les multiples interfaces réseau mises en jeu dans les communications multirails, la bibliothèque doit comparer les attributs NUIOA de chaque réseau et éventuellement en privilégier certains si des conflits d'affinité apparaissent. Nous avons implémenté cette idée dans NEWMADELEINE. En cas de conflit, les performances des différents réseaux peuvent être un bon critère de choix. Une application qui nécessite avant tout une latence faible devrait par exemple être placée près d'une interface QSNET II tandis qu'une application nécessitant un haut débit devrait plutôt être proche d'une interface INFINIBAND. Une application demandant une utilisation intensive du processeur devrait quand à elle être établie sur un processeur libre plutôt qu'à proximité d'une interface réseau. Il est donc nécessaire que l'application puisse donner des indices de placement en fonction de ses besoins. Malheureusement, les interfaces de programmation (MPI, OpenMP, ...) limitent encore les informations pouvant être transmises entre les applications et les bibliothèques logicielles inférieures. Par défaut, nous avons donc fixé le critère de choix sur la bande passante, et laissé aux utilisateurs la possibilité de paramétrer celui-ci. Dans le cas commun d'un multirail homogène avec des réseaux similaires connectés sur différents nœuds NUMA, aucun placement proche d'une interface ou d'une autre ne sera à priori préférable.

4.3 Bilan

Les effets NUMA, connus dans le contexte de l'ordonnancement et du placement des threads et des données, ont un impact notable dans le cadre des transferts sur machines distribuées au travers de réseaux rapides. Nous avons cherché à quantifier ces effets NUIOA (*Non Uniform Input/Output Access*) et observé que si la latence des communications ne souffre que d'un léger surcoût lié à l'éloignement des tâches et de la mémoire des périphériques, le débit des communications peut en être largement affecté. Un placement éloigné entraîne une dégradation de débit visible pour de larges bandes passantes (supérieures à 1 Go/s) et proportionnelle à celles-ci, pouvant atteindre jusqu'à 40% de perte pour des transferts multirails. En pratique, c'est le placement de la mémoire associée à une tâche qui importe, plus que le placement de la tâche communicante elle-même, et ses effets sur le débit apparaissent asymétriques. Ils touchent fortement les zones mémoire cibles et impactent moins les zones mémoire sources, ou inversement selon le matériel employé. De plus, si sur une machine vide la détérioration engendrée par un placement distant n'est pas amplifiée avec l'augmentation de la distance, la charge inhérente à une utilisation applicative réelle est à l'origine de contentions qui peuvent entraver les performances hiérarchiquement.

Nous avons proposé une implémentation de placement automatique dans la plateforme de communications NEWMADELEINE. Lorsqu'un périphérique PCI peut être associé à un descripteur virtuel ou que les pilotes de l'interface fournissent une localité NUMA, les attributs NUIOA sont récupérés, utilisés pour automatiser le placement d'une tâche proche du périphérique, et exposés à l'application. Cette stratégie a été extraite et implémentée dans la bibliothèque `hwloc` ainsi capable d'exposer les affinités NUIOA des interfaces INFINIBAND et MYRI-10G. Notre contribution à la bibliothèque NEWMADELEINE garantit la portabilité des performances en plaçant automatiquement la tâche de communication, et permet d'obtenir des résultats identiques à ceux d'un placement manuel.

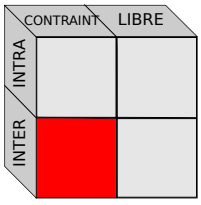
Si ces premiers travaux permettent une prise en charge des effets NUIOA en adaptant le placement des tâches communicantes à la localité des cartes réseau, cette stratégie ne peut convenir à des applications nécessitant un placement prédéfini des tâches au travers de la machine. Celui-ci, requis par exemple pour favoriser l'équilibrage de charge ou les affinités entres les tâches, ne peut ainsi être modifié sans risque de compromettre les besoins applicatifs. Elle ne peut également pas être appliquée aux cas d'applications multithreadées homogènes ou de plusieurs applications exécutées de façon concurrentes, pour lesquels se pose le problème de savoir comment répartir les différentes tâches si elles sont des candidates équivalentes à un placement proche des cartes. Une idée adéquate serait ainsi non pas de chercher à adapter le placement aux contraintes NUIOA, mais d'adapter les stratégies de transfert mises en œuvre dans les bibliothèques de communication à celles-ci.

Chapitre 5

Adaptation des communications MPI aux contraintes NUIOA

Sommaire

5.1 Adaptation des transferts réseau multirails aux effets NUIOA 86
5.1.1 Stratégie de transfert multirail dans OPEN MPI 86
5.1.2 Implémentation 88
5.1.3 Variation du ratio de division et performances des transferts multirails . 90
5.1.4 Bilan 95
5.2 Intégration des contraintes NUIOA dans les opérations collectives MPI . . 95
5.2.1 Collectives hiérarchiques et placement NUIOA 95
5.2.2 Mise en œuvre et performances 100
5.2.3 Bilan 104
5.3 Vers des stratégies de transfert réseau adaptées à la localité des cartes . . . 104



Le rôle notable du standard MPI dans le développement des applications parallèles suscite de nombreux efforts pour le perfectionnement de ses implémentations. L'émergence des architectures multicœurs et la hiérarchisation interne des nœuds de calcul qui en découle constituent un moteur de recherche essentiel au progrès de ces implémentations en termes de performances. Des travaux s'attaquent ainsi au problème de l'impact de la topologie des nœuds de calcul sur les communications intra-nœud, mais son effet sur les communications réseau est encore trop rarement pris en compte au sein des bibliothèques MPI.

Par souci de reproductibilité et de performance, les applications MPI profitent le plus souvent d'un placement des processus prédéfini et parfois au service des affinités applicatives, qu'il serait délicat d'ajuster à la localité des cartes réseau sans en altérer le bénéfice. Adapter le placement des tâches à la position physique des cartes apparaît donc difficile dans ce contexte, et nous nous sommes intéressés à la problématique orthogonale : *adapter les stratégies* de communication invoquées par les bibliothèques aux contraintes NUIOA. Ce chapitre présente deux propositions effectuées dans ce sens. Dans la première, nous nous intéressons à la façon dont un processus donné devrait

utiliser les cartes réseau. Au contraire, notre seconde contribution considère pour une interface réseau donnée la possibilité de choisir quelle tâche devrait l'exploiter prioritairement.

Il est courant que les nœuds de calcul soient dotés de plusieurs interfaces réseau utilisées conjointement pour effectuer des transferts *multirails*. La localité de ces interfaces peut être différente lorsque celles-ci sont interconnectées par des bus d'entrées-sorties distincts, complexifiant les aspects NUIOA associés au placement d'une tâche. Nous proposons ainsi d'adapter la quantité de données envoyées sur chaque "rail" réseau en fonction de la localité des processus relative aux différentes interfaces.

Nous nous intéressons ensuite aux opérations collectives MPI. Pour répondre de façon efficace aux schémas de communication complexes qu'elles mettent en œuvre, de nombreux algorithmes ont été développés pour structurer et minimiser les échanges inter-processus nécessaires. La plupart des bibliothèques MPI proposent ainsi des algorithmes de collectives hiérarchiques pour lesquelles les transferts réseau sont assurés pour l'ensemble du nœud par un processus maître (*leader*) qui est ainsi particulièrement sensible aux aspects NUIOA [1]. Nous proposons ici une sélection du processus maître orientée par la localité NUIOA.

5.1 Adaptation des transferts réseau multirails aux effets NUIOA

Dans les clusters de machines multicœurs équipées de plusieurs cartes réseau, les performances des communications entre les processus dépendent des cœurs sur lesquels les processus sont exécutés, et de leur distance par rapport aux interfaces réseau. Nous proposons dans cette partie une distribution des données des larges messages sur les différentes interfaces, ajustée en fonction de la distance des cartes pour contrebalancer les effets NUIOA et améliorer les performances des communications multirails. Nous présentons les résultats de notre approche implémentée au sein de la bibliothèque OPEN MPI.

5.1.1 Stratégie de transfert multirail dans OPEN MPI

Gérer les affinités à l'intérieur d'une machine NUMA requiert en général de placer les tâches communicant fortement ou nécessitant un grand nombre de synchronisations, au sein d'un même nœud NUMA ou sur des cœurs partageant un cache. Cependant, la répartition de la charge de travail au travers de la machine nécessite l'utilisation de l'ensemble des ressources de calcul et mémoire disponibles. Trouver un bon compromis entre ces contraintes est un problème difficile et dépend des besoins de l'application, comme suggéré en Figure 5.1. Ajouter à cela le problème de localité des interfaces réseau crée de nouvelles exigences alors que certains cœurs ne disposent pas d'un bus d'entrées-sorties local. L'idée vient alors de dédier ces cœurs pour des tâches avec des besoins de communication restreints, et de garder des placements privilégiés proches des périphériques pour des tâches fortement communicantes comme illustré en Figure 5.1 (b).

1. Ce cas s'apparente également au contexte d'applications multithreadées avec un processus par nœud pour lesquelles l'un des threads est affecté au traitement des communications.

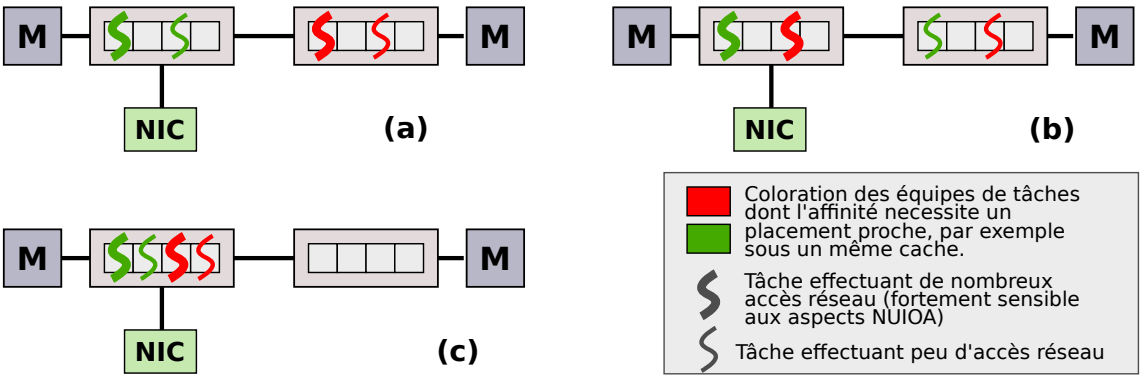


FIGURE 5.1 – *Distribution de 4 tâches sur une architecture bi quadri-cœur.*
(a) Avec une charge répartie sur les processeurs et tenant compte de l'affinité entre les tâches.
(b) Avec une charge répartie sur les processeurs et adaptée aux contraintes NUIOA *mais faite au détriment des affinités entre les tâches.*
(c) Favorisant l'affinité entre les tâches et les aspects NUIOA *mais n'assurant pas la répartition sur les deux processeurs.*

Cependant, déceler les tâches fortement communicantes est un problème potentiellement difficile. De plus, un grand nombre d'applications MPI ont un schéma de communication uniforme, les programmeurs cherchant dans la mesure du possible à éviter le parallélisme irrégulier pour assurer l'utilisation de toutes les ressources de calcul. Alors que placer les tâches communicantes proches des périphériques est un problème complexe, nous nous attachons ici à produire la stratégie inverse et à optimiser les communications pour un placement défini des processus. Un tel placement peut par exemple être explicité au cours de l'initialisation de la bibliothèque MPI pour répondre à des contraintes d'affinités spécifiques à l'application. Nous proposons ainsi d'adapter les primitives de communication MPI dans le but d'exploiter au mieux des interfaces réseau multiples.

Distribuer les segments sur les interfaces en fonction de la localité

Plusieurs implémentations MPI offrent la possibilité d'utiliser de multiples interfaces réseau simultanément. Pour des raisons d'optimisation de débit, les messages de grande taille sont généralement divisés au travers de l'ensemble des *rails*, et rassemblés à la réception (Figure 5.2). Les implémentations telles que OPEN MPI [68] et MPICH2/NEWMADELEINE [67] peuvent utiliser plusieurs interfaces et connexions, et adapter dynamiquement l'utilisation de chacune à leurs performances respectives. Par exemple, sur une architecture disposant de deux cartes INFINIBAND connectées l'une par DDR et l'autre par QDR, un message de 3 Mo pourra être divisé en un segment de 1 Mo envoyé sur le lien DDR et un second segment de 2 Mo transitant sur le lien QDR. Dans le cas d'une configuration homogène, le message sera divisé en segments de tailles égales.

Comme nous l'avons vu dans le Chapitre 4, le débit réel attendu sur les rails réseau ne dépend pas seulement de la configuration réseau mais également de la localisation des processus communicants. Nous proposons ainsi d'ajuster la taille des segments sur chaque interface en fonction de la position du processus émetteur relative à celle des différentes cartes et nommons cette stratégie *multirail hétérosplit*.

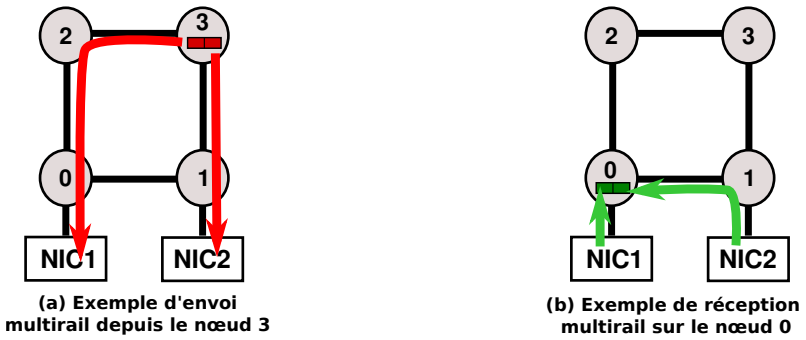


FIGURE 5.2 – *Exemple de transferts multirails, (a) en envoi, (b) en réception.*

5.1.2 Implémentation

Nous avons implémenté cette idée dans la version 1.4.1 de la bibliothèque OPEN MPI. La conception de cette implémentation MPI est basée sur une architecture par composants qui assure sa portabilité et son extensibilité [70, 68, 69]. Le MCA (*Modular Component Architecture*), squelette de l'architecture, offre une interface simple permettant d'ajuster l'environnement d'exécution, et est composé de "*Component Frameworks*". Ces derniers assurent la gestion d'un où plusieurs modules regroupés par domaine de fonctionnalités. Ils sont responsables de découvrir, charger, utiliser et fermer les modules requis par les paramètres de la bibliothèque.

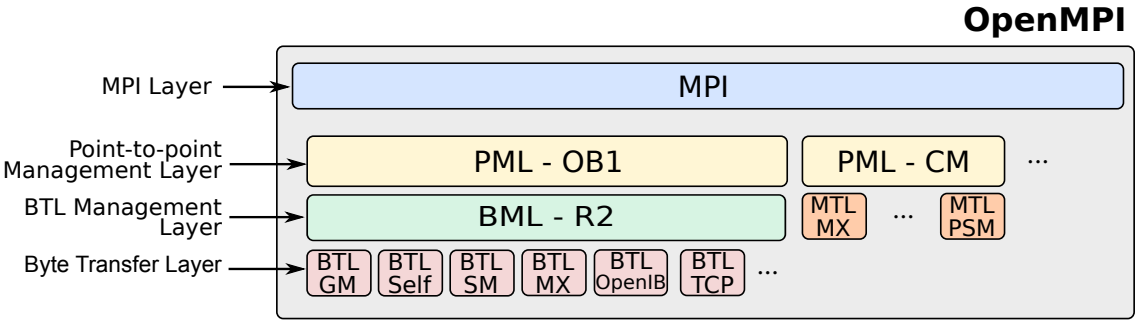


FIGURE 5.3 – *Représentation partielle de la pile logicielle* OPEN MPI *ciblée sur les composants responsables des transferts point-à-point.*

La conception et l'implémentation d'un transfert point-à-point OPEN MPI repose sur l'utilisation des composants : *Point-to-point Management Layer* (PML) ; *Byte Transfer Layer* (BTL) et *BTL Managment Layer* (BML) [73] au-dessous de la couche MPI (Figure 5.3). Chacun d'entre eux définie une interface et assure un cloisonnement fonctionnel. Tandis que le PML implémente la sémantique d'un protocole de communication point-à-point donné et administre la livraison des messages dans leur globalité, le BTL manipule les transferts de données au travers du réseau sans avoir connaissance du protocole de communication utilisé aux niveaux supérieurs. Le composant BML fournit quant à lui un ensemble de services au démarrage des tâches et assure la découverte et le maintien de l'ensemble des BTLs pouvant être utilisés.

La Figure 5.4 illustre les étapes intervenant lors d'un transfert. Chaque interface réseau est gérée par un composant BTL qui collecte les différentes bandes passantes attendues en fonction du modèle et de la configuration **(1)**. Par défaut, ces bandes passantes sont consultées dans le BML qui calcule ensuite un "poids" pour chaque BTL **(2)**. Envoyer un message de taille conséquente en multirail consiste alors à envoyer un morceau par BTL **(4)** avec un *ratio de division* **(3)** déterminé de sorte que la taille du tronçon soit proportionnelle au poids du BTL.

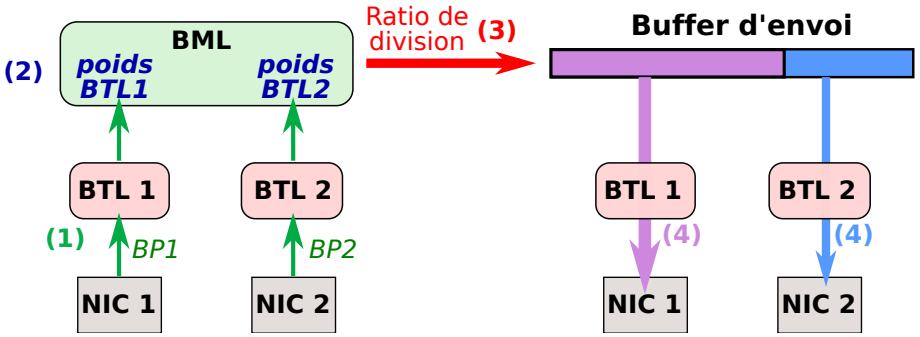


FIGURE 5.4 – *Transfert multirail au travers de la pile* OPEN MPI.
(1) Collecte des informations de bande passante (BP) pour chaque BTL.
(2) Calcul du poids de chaque BTL par le BML.
(3) Définition du ratio de division proportionnel aux poids.
(4) Répartition des données envoyées sur chaque carte au travers des BTLs en fonction du ratio.

Nous avons modifié le composant BML R2 pour prendre en compte la localité NUIOA dans le calcul de ces poids. La bande passante attendue pour chaque BTL est ajustée en regardant les positions physiques du BTL et du processus courant. Cette phase est effectuée assez tard dans l'initialisation et notamment après que le placement des processus sur les unités de calcul ait été pris en charge par le composant OPEN MPI qui en est responsable. Il en résulte ainsi un ratio de division ajusté aux contraintes NUIOA et propre à chaque instance de BML employée par les différents processus.

Le poids du composant BTL modifié exploite des informations de placement des interfaces réseau sur la topologie sous-jacente. OPEN MPI a récemment intégré l'utilisation de la bibliothèque `hwloc` qui offre des fonctionnalités de placement et comme nous l'avons expliqué au chapitre précédent (section 4.2.1), une détection de l'emplacement physique des cartes lorsque les *devices PCI* peuvent être associés aux descripteurs logiciels correspondants. Au travers de cette bibliothèque, OPEN MPI est ainsi capable d'assurer le placement des processus sur la topologie matérielle et la détection des cartes réseau, et peut ainsi transmettre ces informations au BML.

La version 1.4.1 sur laquelle nous avons mené notre étude ne disposant pas encore de cette bibliothèque, nous y avons ajouté une détection des cartes "bas niveau" de la même façon que nous l'avions effectuée pour les travaux sur la bibliothèque NEWMADELEINE antérieurs à la création de `hwloc`, et laissé le module `paffinity`, en charge le l'affinité processeur, fixer le placement des processus. Cette implémentation temporaire et peu élégante a bien sûr vocation à être remplacée par son homologue au travers d'`hwloc` dès la prochaine version stable d'OPEN MPI.

5.1.3 Variation du ratio de division et performances des transferts multirails

Effet NUIOA sur notre plateforme de test

Nous avons évalué les répercussions des multiples attraits NUIOA sur les performances des communications réseau multirails sur le cluster Borderline (mentionné en Section 4.1.3 et détaillé en Annexe A.4) de la plateforme expérimentale Grid'5000 [6]. Ce cluster est composé d'une dizaine de machines quatre nœuds NUMA constitués de processeurs bi-cœurs AMD OPTERON 8218. Chacune d'entre elle possède deux interfaces réseau, homogènes ou hétérogènes selon la configuration avec 2 cartes INFINIBAND, 2 cartes MYRI-10G ou une carte de chaque technologie. Les interfaces réseau sont attachées l'une sur le nœud NUMA 0, l'autre sur le nœud 1 (Figure 5.5). Un processus peut ainsi être proche d'une carte réseau et à distance de la seconde (placement sur les nœuds 0 et 1), ou à distance des deux cartes (placement sur les nœuds 2 et 3).

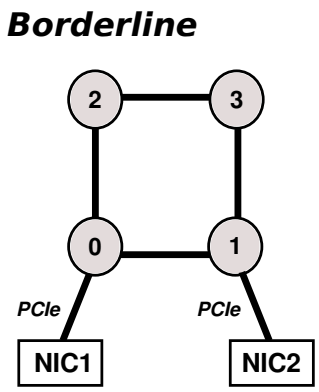


FIGURE 5.5 – *Topologie NUMA des machines Borderline.*

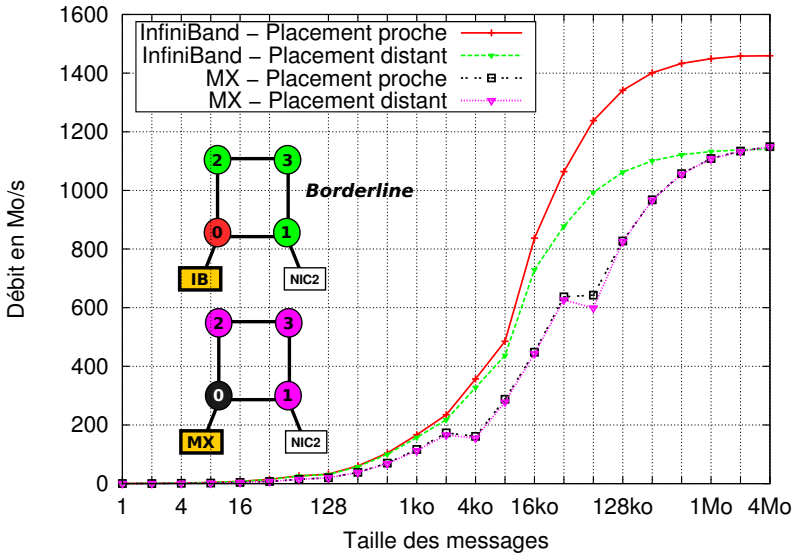


FIGURE 5.6 – *Influence de la localité des processus et des cartes réseau sur le débit d'un ping-pong monorail avec* OPEN MPI *sur la plateforme Borderline.*

Les caractéristiques NUIOA observées sur ces architectures (section 4.1.3) pour des ping-pong monorails au-dessus de l'implémentation OPEN MPI sont rappelées en Figure 5.6. Elles montrent des effets NUIOA mineurs avec l'utilisation du réseau MYRI-10G mais importants sur le débit des communications sur INFINIBAND avec une variation supérieure à 20%.

Ping-pong multirail avec OPEN MPI

Nous avons étudié le potentiel de notre stratégie en faisant varier manuellement le pourcentage de données envoyées sur chaque carte pour des tests de ping-pong IMB [31]. La Figure 5.7 présente les débits obtenus pour des transferts multirails de messages de 1 Mo, en fonction du placement des processus et pour une configuration homogène avec deux cartes INFINIBAND.

Ces résultats indiquent clairement que pour un processus placé proche d'une carte (nœud NUMA 0

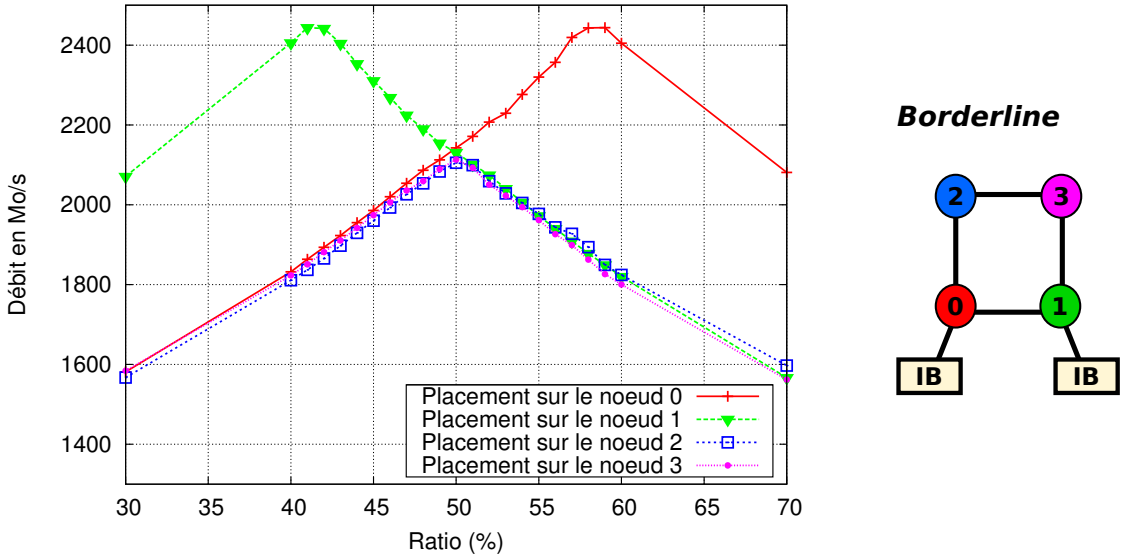


FIGURE 5.7 – *Débits de ping-pong IMB multirails pour des messages de 1 Mo en fonction du ratio de division et du placement des processus. Le ratio correspond au pourcentage de données envoyées par la carte attachée au nœud 0.*

ou 1), l'implémentation MPI doit privilégier la carte locale. Le débit optimal est alors obtenu lorsque l'interface proche prend en charge 58% de la taille du message à transférer, laissant un pourcentage moindre à la charge de la seconde carte. En utilisant une telle mise au point plutôt que la stratégie *isosplit* qui consiste à diviser le message en deux parts égales attribuées à chaque carte [2], les communications profitent d'un gain notable de débit de 15%. Lorsque le processus est distant des cartes (nœuds 2 et 3) la stratégie isosplit est le choix le plus intéressant.

Le ratio expérimental de 58% est assez proche du quotient du débit monorail optimal sur la somme des débits de chaque rail, dérivé de la Figure 5.6. En effet, le débit monorail d'un placement proche atteint 1460 Mo/s tandis que celui relatif au placement distant n'excède pas 1140 Mo/s. Si l'on considère une bande passante multirail "théorique" comme la somme des débits monorails sur chaque carte, soit au total 2600 Mo/s, le débit sur la carte locale représente alors 56% de ce total (Figure 5.8 (a)). Dans le cas d'un placement distant, le débit des transferts monorails ne varie pas entre les différents placements éloignés des cartes (Section 4.1.3). Les débits monorails sur les deux cartes sont donc équivalents comme illustré Figure 5.8 (b). Chacun représente donc 50% du débit somme (bande passante multirail théorique), soit la valeur du ratio de division observé pour les placements distants.

Les tests similaires effectués avec une configuration réseau homogène appuyée sur MYRI-10G, traduisent un effet bien plus faible de la modification du ratio de division, avec un débit maximum obtenu lorsque 51% des données sont envoyées sur la carte locale et un gain peu significatif. Ce résultat n'a rien d'étonnant si l'on considère le très faible impact NUIOA observé sur le débit monorail au travers de cette technologie réseau (Figure 5.6). L'utilisation hétérogène des réseaux

2. Stratégie utilisée par défaut par OPEN MPI lorsque la configuration réseau est homogène.

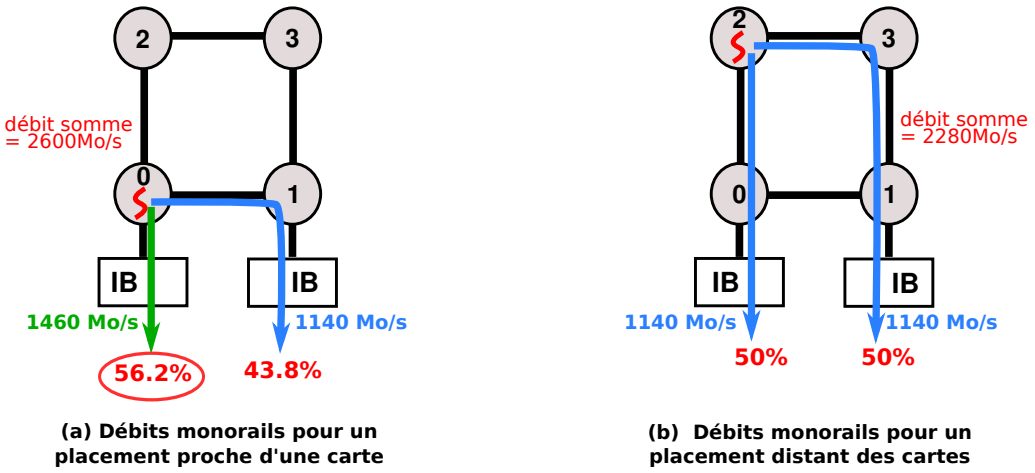


FIGURE 5.8 – *Ratio de division multirails dérivé des débits monorails relatifs au placement.*
(a) Pour un placement proche, le pourcentage de données envoyées sur la carte locale est majoritaire (56% de la somme des données envoyées sur chaque carte).
(b) Pour un placement distant, le pourcentage de données envoyées sur chaque carte est identique.

MYRI-10G et INFINIBAND concorde avec nos attentes avec une variation intermédiaire : lorsque le processus est placé proche de la carte INFINIBAND, celle-ci doit être favorisée de façon assez significative (57%) ; tandis que pour un placement proche de l'interface MYRI-10G, la carte locale doit être avantagée de façon plus ténue (51%). Pour toutes les configurations, la stratégie isosplit reste la plus favorable lorsque les processus sont distants des deux cartes. L'ensemble de ces résultats confirment que la combinaison des débits des transferts monorails pour un placement donné est une bonne méthode pour approximer le ratio de division optimal des communications point-à-point multirails.

Effet de la contention

La seconde étape de notre analyse considère l'impact des contentions sur les bus mémoire sur nos communications multirails. En effet, les résultats exposés précédemment ont été mesurés sur des machines inoccupées sur lesquelles les transferts entre les nœuds NUMA et les cartes réseau ne sont perturbés par aucun transfert concurrent sur les liens. Dans le cas d'applications réelles, les processeurs peuvent accéder à de la mémoire distante, se synchroniser ou échanger des données, générant du trafic sur les liens HYPERTRANSPORT de notre architecture. Pour en étudier les répercussions, nous avons ajouté des accès distants à la mémoire depuis divers processeurs et concurrents à nos tests de ping-pong multirails.

Nous attendions ici une détérioration des transferts vers les cartes distantes augmentant le poids de la carte locale [3]. Les résultats de ces tests traduisent une réduction de la bande passante *générale*, mais ne modifient en rien la valeur du ratio de division optimal. Ce résultat est surprenant car nous avons pris soin de générer des contentions sur les liens parcourus durant le transfert des données vers les cartes distantes, extraits des tables de routage HYPERTRANSPORT. Pour exemple,

3. Ou lorsque le processus communicant n'est proche d'aucune carte, une éventuelle augmentation du poids de la carte la "moins éloignée".

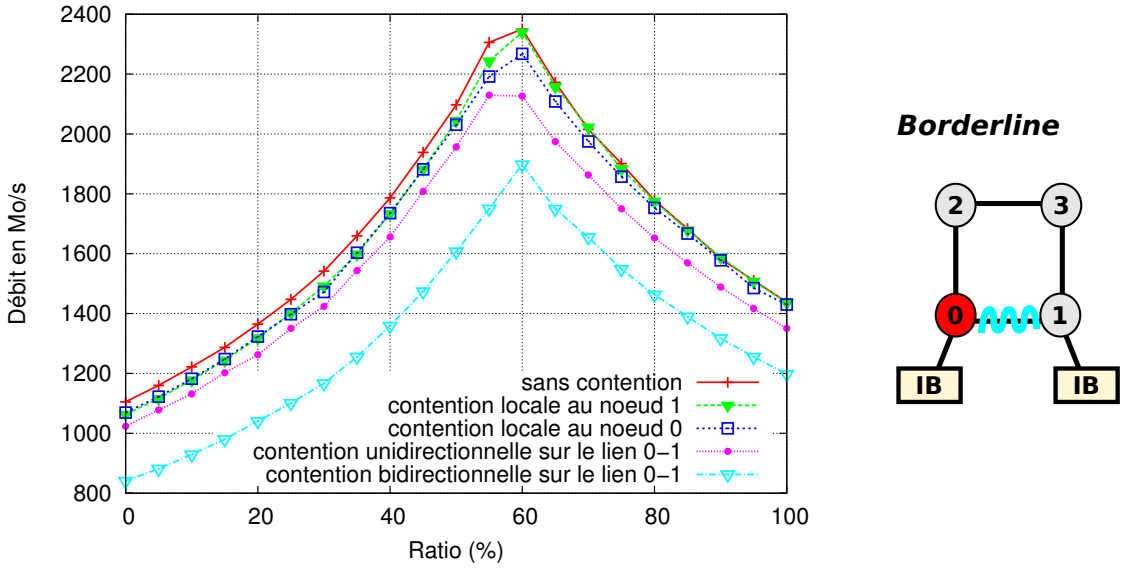


FIGURE 5.9 – *Débits de ping-pong IMB multirails sur* INFINIBAND*, effectués depuis le nœud 0 pour des messages de 1 Mo en fonction de ratio de division et soumis à diverses contentions.*

la Figure 5.9 présente les débits des transferts multirails de 1 Mo depuis le nœud NUMA 0 en fonction du ratio de division utilisé. Les différentes contentions générées au cours de nos tests, locales aux nœuds 0 ou 1, ou sur le lien traversé pour atteindre la carte distante dégradent plus ou moins sévèrement les performances. Dans chaque cas, le débit maximum observé est obtenu pour un ratio constant de 58%.

L'absence de fluctuation du ratio de division optimum traduit ainsi une influence négative des contentions sur l'ensemble de la bande passante mémoire plutôt que sur la bande passante seule du lien congestionné.

Opérations collectives

Nous nous intéressons maintenant au cas plus général des opérations collectives MPI. Puisque les processus communiquent désormais simultanément depuis l'ensemble des nœuds NUMA de la machine, nous étudions le ratio à utiliser pour chaque processus en fonction de sa localisation NUMA et de la position physique des cartes.

La Figure 5.10 présente les performances d'un test de communication *all-to-all* entre deux machines *Borderline* avec une configuration réseau INFINIBAND homogène, en fonction du ratio de division pour chaque placement de processus. Nous distinguons ici les processus proches d'une carte réseau (sur les nœuds 0 ou 1) ou éloignés de chacune (sur les nœuds 2 ou 3).

Ces résultats révèlent tout d'abord que le ratio le plus intéressant pour les processus qui ne sont proches d'aucune carte est de 50%. Ce constat correspond à nos remarques faites sur les communications point-à-point : dans le cas de placement distants, les effets NUIOA ne prennent pas plus d'ampleur avec l'augmentation de la distance aux périphériques réseau, aucun d'entre eux n'est donc à privilégier. Une caractéristique plus intéressante est que les processus proches d'une carte

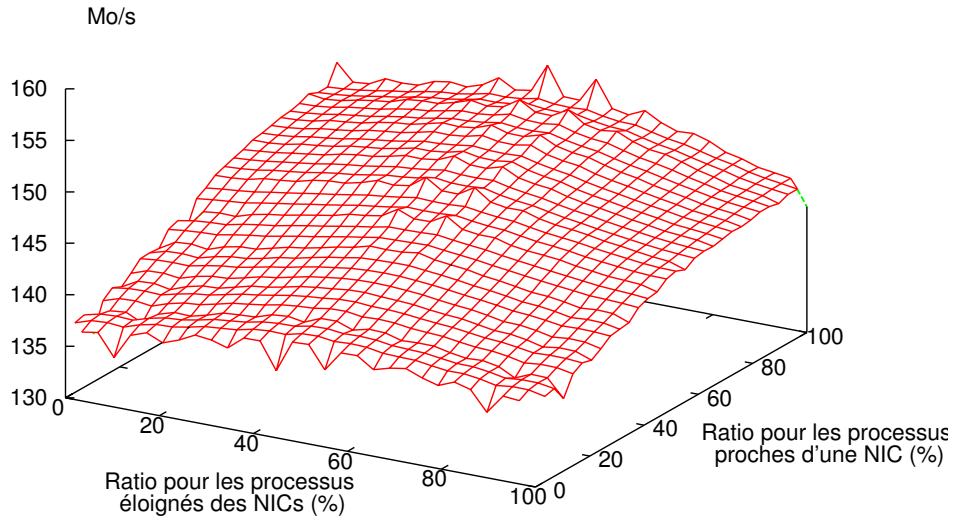


FIGURE 5.10 – *IMB all-to-all, débit par processus entre 16 processus sur 2 hôtes disposant de 2 cartes* INFINIBAND*, en fonction du ratio de division.*

ne devraient exploiter que celle-ci (ratio de 100% pour la carte proche) plutôt que de diviser le message en plusieurs tronçons. Cette constatation contredit les observations faites sur l'ajout de contention dans nos tests point-à-point, puisque les congestions occasionnées jouent maintenant un rôle critique sur les performances qui requiert de revaloriser la carte locale. Nous pensons que les contentions générées dans le test *all-to-all* sont beaucoup plus importantes que celles mises en œuvre dans la section précédente, expliquant ainsi les contrastes entre les ratios optimaux.

Concrètement, sur nos hôtes équipés de cartes INFINIBAND, l'ajustement du pourcentage de données envoyées sur chaque carte et pour chaque processus [4] apporte, comparé à la stratégie isosplit usuelle, un gain de 5% sur le débit des communications all-to-all. L'utilisation conjointe des réseaux INFINIBAND et MYRI-10G suggère une répartition des données équivalentes avec un gain de l'ordre de 2%. Dans le cas d'une configuration homogène avec le réseau MYRI-10G, les conséquences de la distribution des données sur les périphériques apparaissent insignifiantes (les écarts de performance entre plusieurs exécutions sont supérieurs aux variations induites par le facteur de division).

Nous avons étendu notre étude à diverses opérations collectives et constaté pour l'opération *all-gather* des ratios adéquats équivalents à ceux du *all-to-all*, avec un gain potentiel de 3% de débit avec l'utilisation de 2 cartes INFINIBAND. Sur les autres opérations qui nécessitent un nombre de communication inter-processus moindre, nos expérimentations n'exhibent pas de fluctuations relatives au paramètre de division choisi.

4. 100% des données envoyées sur la carte locale, ou 50% sur chaque interface si le processus est distant de chacune.

5.1.4 Bilan

Pour assurer le passage à l'échelle des communications réseau, les machines de calcul sont généralement équipées de multiples interfaces réseau. La généralisation des architectures NUMA dans les clusters de calcul s'accompagne d'accès non uniformes à ces interfaces dont les performances sont liées à leur localisation physique et à la position relative des processus communicants.

Nous avons présenté une optimisation permettant d'adapter l'utilisation de multiples interfaces réseau en fonction de cette localité. Nos résultats indiquent que dans le cadre de communications point-à-point, une division intelligente des larges messages entre les interfaces peut être dérivée des débits monorails respectifs. Elle permet ainsi de contrebalancer les effets NUIOA significatifs et offre un gain de débit pouvant atteindre 15%. Pour des schémas de communication intensifs tels que les opérations "all-to-all", les expériences révèlent que les processus proches d'une carte devraient utiliser celle-ci de façon exclusive pour éviter les contentions sur les bus mémoire et optimiser les performances de transfert. Nous avons ainsi implémenté au sein de la bibliothèque OPEN MPI une stratégie de transfert multirail *hétérosplit* qui ajuste la quantité de données envoyées sur les cartes réseau par chaque processus communicant. Elle s'appuie pour cela sur le placement des processus relatif à la localité des périphériques qui peut être extraite au travers de la bibliothèque `hwloc`.

5.2 Intégration des contraintes NUIOA dans les opérations collectives MPI

Nous proposons dans cette section de prendre en compte les aspects NUIOA au sein des opérations collectives dont l'efficacité et le passage à l'échelle sont déterminants sur les performances des applications MPI. Comme nous l'avons expliqué en Section 2.2.2, les opérations collectives ont profité de nombreux perfectionnements visant à réduire le trafic interprocessus. Les implémentations naïves ont ainsi été remplacées par des communications structurées appuyées par exemple sur des arbres de transfert. Celles-ci réduisent le nombre d'échanges nécessaires entre les nœuds des grappes et favorisent le passage à l'échelle inhérent à la multiplication des nœuds dans ces structures. Avec la multiplication des unités de calcul au sein des machines, l'augmentation des échanges entre les processus locaux a orienté les recherches vers le perfectionnement des transferts intra-nœud, et une prise en compte de la localité dans l'organisation des échanges. Les algorithmes mis en place favorisent ainsi les transferts intra-nœud et tendent à réduire au maximum les communications réseau plus coûteuses. L'idée introduite dans cette section est d'intégrer à l'implémentation des opérations collectives la notion de localité NUIOA pour favoriser l'efficacité des transferts réseau inévitables qui interviennent au cours de ces opérations.

5.2.1 Collectives hiérarchiques et placement NUIOA

Les opérations collectives se décomposent en multiples transferts effectués en mémoire partagée ou au travers du réseau. Dans les bibliothèques MPI contemporaines, les implémentations de ces types de communication sont généralement distinctes et peuvent être combinées de façon

hiérarchique en fonction de la localité. Par exemple, une opération de diffusion (*broadcast*) peut être divisée en plusieurs étapes : le processus racine de l'opération envoie les données au travers du réseau sur chaque nœud, après quoi celles-ci sont diffusées aux processus locaux en mémoire partagée.

Les collectives hiérarchiques s'appuient généralement sur l'élection d'un processus maître (*leader*) par nœud responsable de représenter l'ensemble des processus locaux durant l'opération. Ce processus est chargé d'effectuer les envois sur le réseau pour l'ensemble des processus qu'il représente, et est donc particulièrement sensible aux effets NUIOA. Nous proposons de modifier l'élection du leader en fonction de la localité des processus et des cartes réseau, de sorte à choisir un processus disposant de par son placement d'un accès privilégié au réseau. Outre l'amélioration des performances de transfert que ce choix peut entraîner, cette stratégie permet également de réduire la contention sur les liens mémoire en évitant un transfert distant comme illustré par la Figure 5.11.

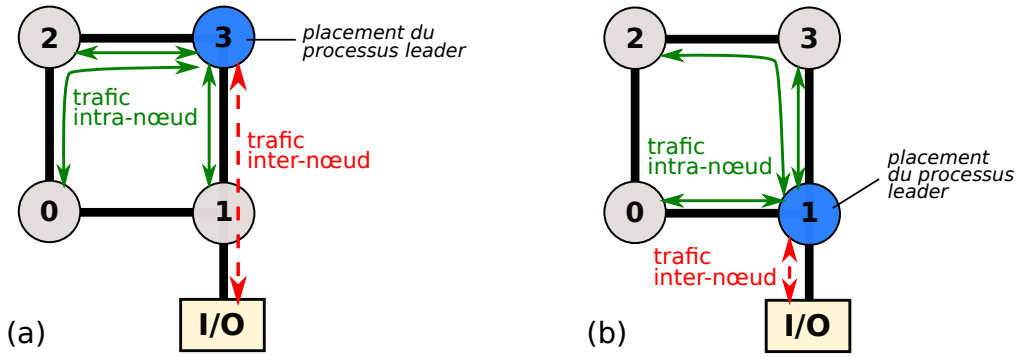


FIGURE 5.11 – *Trafic sur les liens mémoire d'une machine NUMA 4 nœuds :*
(a) Pour un placement du leader distant de la carte réseau (nœud 3). Des contentions peuvent apparaître sur le lien entre les nœuds 1 et 3 emprunté pour des transferts intra et inter-nœuds.
(b) Pour un placement du leader proche de l'interface. Les transferts inter-nœuds ne génèrent pas de trafic sur les liens mémoire utilisés par les échanges intra-nœud.

Notre étude des effets NUIOA (Chapitre 4.1) nous a mené à la conclusion que l'impact d'un placement distant de l'interface réseau est faible sur la latence (de l'ordre d'une centaine de nanosecondes par lien mémoire traversé) mais notable sur les débits élevés. Ils touchent ainsi principalement le transfert de très larges messages. De plus, ces effets apparaissent asymétriques et importent selon les plateformes sur les zones sources ou sur les zones destinations des données, bien que ce second cas semble à ce jour plus commun. Notre stratégie d'élection des processus maîtres devraient ainsi se concentrer sur les processus qui envoient ou réceptionnent de larges messages, et présenter un intérêt fonction du cas traité et de la plateforme ciblée.

Potentiel du placement NUIOA du leader dans les opérations collectives

Pour prédire le potentiel de notre proposition, il est important de considérer les schémas de communication invoqués par les opérations collectives. Comme nous l'avons évoqué en Section 2.2.2, ceux-ci peuvent faire intervenir un émetteur et de multiples récepteurs (*one-to-all*), de multiples émetteurs pour un récepteur (*all-to-one*), ou de multiples émetteurs et récepteurs (*all-to-all*).

Dans le cas des opérations one-to-all telles que le *broadcast*, le processus racine à l'origine des

transferts transmet les données aux autres processus sans recevoir de données en retour. Il est désigné de façon naturelle comme le processus leader pour son propre nœud. Il ne peut donc pas être sélectionné par notre stratégie [5] et est ainsi susceptible d'envoyer des messages réseau tout en se trouvant loin des interfaces. Au contraire, la sélection de l'ensemble des processus réceptionnant les données sur les nœuds distants (les multiples autres leaders) peut faire l'objet de soins particuliers pour éviter les effets NUIOA. En les choisissant proches des cartes, les performances devraient être améliorées sur les architectures dont le placement des zones *destinations* des données importe.

Pour les opérations de type all-to-one telles que l'opération de rassemblement (*gather*), c'est le processus racine qui est la cible des transferts de données. Encore une fois, les contraintes NUIOA ne peuvent être prises en compte pour cette racine sans altérer la répartition prédéfinie des processus. Sur les nœuds distants, la sélection des leaders proches de la carte réseau peut être mise à contribution, mais n'est susceptible de bénéficier aux performances que sur les architectures pour lesquelles le placement des données *sources* importe majoritairement.

Pour les opérations one-to-all et all-to-one, une solution permettant de considérer les effets NUIOA sur le nœud racine de l'opération serait d'ajouter un processus leader proche de l'interface réseau et chargé de faire l'intermédiaire entre la racine et le périphérique. Cette étape additionnelle pourrait toutefois nuire aux performances globales et remettrait en cause le modèle hiérarchique considéré dans notre étude.

Les opérations all-to-all telles que le *allgather* [6] ne disposent pas d'un processus racine particulier mais peuvent être implémentées en utilisant de multiples leaders en charge de regrouper, échanger et distribuer les données de l'ensemble des processus locaux. Chacun de ces processus peut alors être choisi pour son placement proche de la carte et faire bénéficier l'ensemble des performances d'une gestion NUIOA intelligente.

	Placement sur le nœud racine	Placement des autres leaders
one-to-all	Processus racine émetteur des données, **adaptation NUIOA non envisageable.**	Leaders distants récepteurs des données, **bénéfice si les zones *destinations* importent.**
all-to-one	Racine réceptrice des données, **adaptation NUIOA non envisageable.**	Leaders distants sources des données, **bénéfice si les zones *sources* importent.**
all-to-all	Données émises et reçues depuis l'ensemble des nœuds, **bénéfice attendu pour chaque placement NUIOA des leaders sur les nœuds** .	

TABLE 5.1 – *Résumé de l'impact du placement* NUIOA *des leaders attendu dans le cadre de notre contribution en fonction des types d'opérations collectives et des architectures, sensibles au placement des zones mémoire sources ou bien destinations des échanges.*

Performances des opérations collectives en fonction du placement des processus

Nous proposons ici une évaluation rapide des effets NUIOA sur les opérations collectives *broadcast* (one-to-all) et *gather* (all-to-one) effectuées au sein de la bibliothèque OPEN MPI pour valider

5. Dans le cadre de notre contribution, le placement de ce processus peut avoir été défini pour servir les performances applicatives globales. De plus, les opérations collectives invoquées au cours de l'application peuvent utiliser des racines différentes. L'adaptation de leur placement nécessiterait des migrations de processus dont le coût pourrait être supérieur au bénéfice apporté par un placement NUIOA.

6. Opération de multi-diffusion.

nos attentes quant à l'apport potentiel de notre stratégie.

La plateforme employée pour nos tests comprend la grappe de calcul Borderline (Annexe A.4), précédemment utilisée pour nos expérimentations sur les transferts multirails. Notre étude sur les opérations collectives se détache cependant de ces cas particuliers de transfert réseau. Dans les expériences qui suivent, seule une carte INFINIBAND (choisie pour sa sensibilité aux aspects NUIOA) est ainsi mise à contribution dans nos échanges réseau. Pour rappel [7], les effets NUIOA observés sur ces architectures pour des tests de ping-pong au-dessus de la bibliothèque OPEN MPI sont caractérisés par une dégradation de débit de 20% pour un placement distant de la carte. L'asymétrie observée sur ces machines témoigne d'un impact sur le seul placement des zones mémoire dans lesquelles les données sont déposées [8].

Nous avons également mené des expériences sur le cluster *Mirage* (Annexe A.5). Sur ces architectures disposant de cartes INFINIBAND QDR qui assurent de très larges bandes passantes réseau (pouvant atteindre plus de 2300 Mo/s pour des transferts point-à-point au dessus de OPEN MPI), un placement distant des processus engendre une perte de débit supérieure à 40%. Les machines *Mirage* sont également caractérisées par un impact NUIOA touchant particulièrement les zones mémoire sources des échanges, au contraire des machines du cluster Borderline.

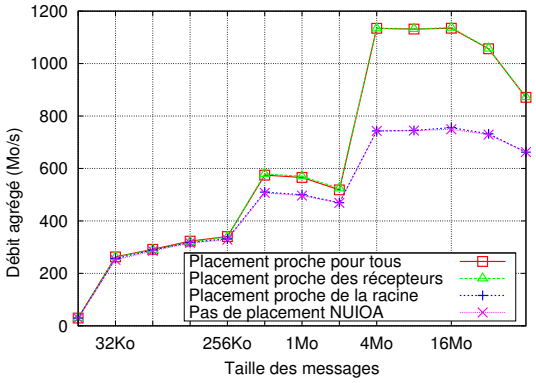


FIGURE 5.12 – *Performances du test IMB Bcast effectué entre 8 nœuds sur le cluster Borderline (8 processus, 1 par nœud).*

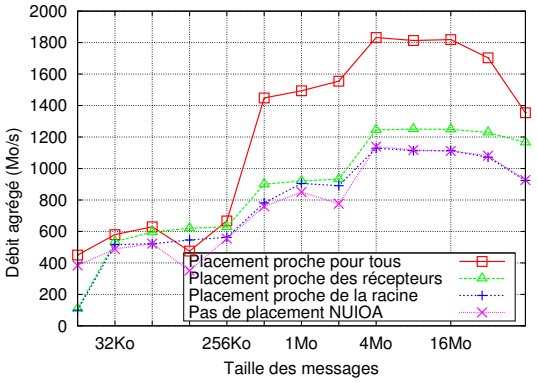


FIGURE 5.13 – *Performances du test IMB Bcast effectué entre 8 nœuds sur le cluster Mirage (8 processus, 1 par nœud).*

Les Figures 5.12 et 5.13 présentent les débits agrégés [9] du test IMB Bcast entre 8 processus répartis sur différents nœuds des grappes de calcul Borderline et Mirage (1 unique processus par nœud). Chaque transfert intervenant au cours de l'opération de broadcast est ainsi effectué au travers du réseau. Nous considérons ici le placement du processus racine de l'opération et des processus récepteurs des données, proche ou distant de l'interface réseau employée. Sur les machines Borderline, nous observons un gain de performance (30%) dès lors que les processus récepteurs sont à proximité des cartes. Pour les opérations one-to-all, notre stratégie d'élection des leaders sur les nœuds distants devrait ainsi profiter aux performances de transfert inter-nœuds sur ces architec-

7. L'étude présentée en Chapitre 4.1 inclut le détail des effets NUIOA observés sur les plateformes Borderline et Mirage.
8. Zones mémoire destinations.
9. Calculés à partir du total des données transférées au cours de l'opération.

tures. Sur les machines Mirage, pour lesquelles le débit souffre moins du placement des zones destinations que des zones sources, le placement proche de la racine de l'opération est indispensable à l'obtention de performances maximales. Comme en témoigne la Figure 5.13, un placement soigné des processus récepteurs favorise cependant les performances avec un gain de 12% pour le transfert de larges messages. Pour les échanges one-to-all, cette plateforme devrait ainsi bénéficier de façon moindre mais visible de notre stratégie.

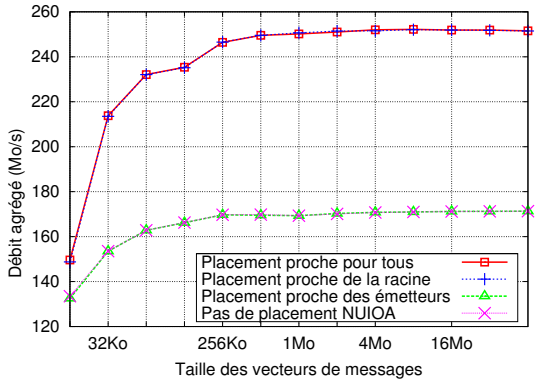


FIGURE 5.14 – *Performances du test IMB Gather effectué entre les 8 nœuds du cluster Borderline (8 processus, 1 par nœud).*

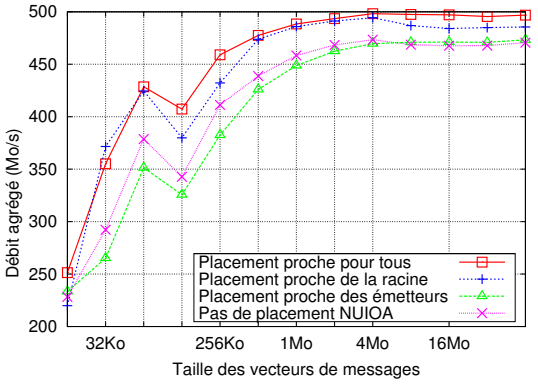


FIGURE 5.15 – *Performances du test IMB Gather effectué entre 8 nœuds du cluster Mirage (8 processus, 1 par nœud).*

Les Figures 5.14 et 5.15 présentent pour leur part les résultats obtenus pour l'opération gather (all-to-one). Sur Borderline seul le placement de la racine importe. La modification des leaders sur les nœuds distants ne devrait donc pas améliorer les performances des transferts inter-nœuds. En théorie, le placement des processus non-racines (émetteurs des données) devraient profiter aux transferts réseau entre les machines Mirage puisque celles-ci sont sensibles au placement des zones mémoire sources. Les résultats traduisent cependant un effet minime de ce placement comparé à celui du processus racine. Nous pensons que cela est dû à des contentions liées au fait qu'un seul processus réceptionne ici les données pour de multiples émetteurs. En conséquence, cette opération apparaît comme une candidate moins intéressante pour notre proposition.

	Cluster Borderline (placement destination crucial)	Cluster Mirage (placement source dominant)
Broadcast	**processus racine** : pas d'effet du placement NUIOA **non-racines** : jusqu'à 30% de gain si placement NUIOA	**processus non-racines** : jusqu'à 12% de gain **racine** : entre 14 et 38% de gain supplémentaire après 512 ko
	⇒ **Avantage notable obtenu avec un placement NUIOA des processus non-racines**	
Gather	**processus racine** : jusqu'à +32% si placement NUIOA **non-racines** : pas d'amélioration	**processus racine** : jusqu'à +5% **non-racines** : effet mineur
	⇒ **Intérêt limité du placement NUIOA des processus non-racines**	

TABLE 5.2 – *Résumé de l'impact du placement des processus sur les transferts réseau pour des opérations collectives avec un processus par nœud sur les grappes Mirage et Borderline.*

5.2.2 Mise en œuvre et performances

Nous avons implémenté notre idée au sein de la bibliothèque OPEN MPI 1.5. La pile logicielle de cette implémentation MPI dispose de plusieurs composants dédiés aux opérations collectives. Ceux-ci sont regroupés dans le module *coll* et proposent divers algorithmes pour les différentes opérations. Le composant employé par défaut dans cette bibliothèque est le module *Tuned*, qui alterne entre différents algorithmes (pipeline, arbre binaire, etc.) en fonction de la taille des messages, et du nombre et de la position des processus. Nos travaux ciblent toutefois un second module nommé *Hierarch* qui implémente un ensemble d'opérations collectives de manière hiérarchique. En effet, les algorithmes mis en œuvre dans le module Tuned bénéficient de nombreuses optimisations peu portables qui réduisent la marge de manœuvre d'étude possible et complexifient l'évaluation de notre stratégie. Au contraire, la relative simplicité des algorithmes du module Hierarch nous semble particulièrement adaptée à l'étude des aspects liés à la localité matérielle.

Ce module utilise de multiples stratégies de transfert et décompose chaque opération en diverses étapes en fonction des niveaux de topologie de la grappe. Il combine entre autre de façon simple les étapes en mémoire partagée pour les communications intra-nœud et les étapes de communication réseau. Pour l'opération de broadcast, l'algorithme consiste à la transmission réseau des données au leader de chaque nœud distant, à la suite de laquelle chacun d'entre eux effectue les transferts en mémoire partagée aux différents processus locaux.

Par défaut, les processus intermédiaires pour chaque nœud sont élus de telle sorte que leur rang local sur le nœud soit équivalent au rang local du processus racine sur le sien (Figure 5.16). Comme dépeint par la Figure 5.17, nous avons modifié cette règle pour choisir en tant que leader un processus placé proche d'une interface réseau.

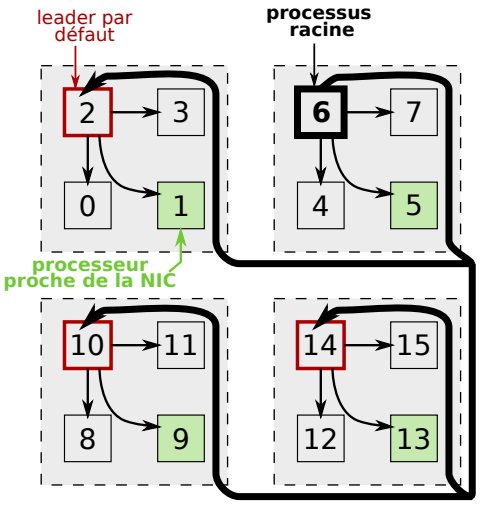


FIGURE 5.16 – *Schéma de communication pour l'opération broadcast du module Hierarch entre 16 processus sur 4 nœuds. Lorsque le processus 6 est racine de l'opération, les processus leaders 2, 10 et 14 désignés par défaut sur les nœuds distants sont éloignés des cartes réseau.*

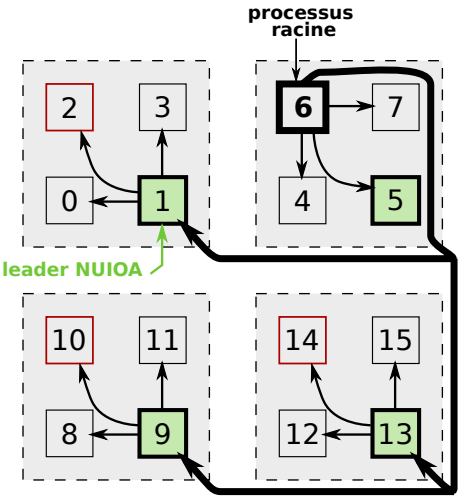


FIGURE 5.17 – *Schéma de communication de notre variante* NUIOA *pour l'opération de broadcast hiérarchique entre 16 processus sur 4 nœuds. Les processus leaders 1, 9 et 13 sont élus par notre stratégie pour leur proximité avec l'interface réseau.*

Cette élection est facilement implémentée au travers de la bibliothèque `hwloc`, qui est non seulement capable de trouver l'emplacement des cartes réseau, mais aussi de retourner directement un ensemble de cœurs proches de celles-ci. Combinée à sa capacité de placement, cette bibliothèque dispose ainsi d'une connaissance complète de l'affinité NUIOA des processus utilisés.

Le module Hierarch ne disposant pas encore de toutes les opérations collectives, seule l'opération de broadcast a pour l'instant pu être concrètement modifiée. Nous pensons cependant que l'extension de ces travaux à d'autres collectives hiérarchiques devrait avantager les performances des transferts réseau.

Performance du broadcast hiérarchique NUIOA

Nous proposons ici les résultats du test IMB Bcast utilisant un processus par cœur. Les leaders locaux jouent ainsi leur rôle d'intermédiaire au sein de l'algorithme hiérarchique. La Figure 5.18 présente les débits obtenus sur la grappe Borderline en fonction de la stratégie d'élection de processus leaders (par défaut, ou grâce à notre élection NUIOA). Pour cette opération, nous proposons les résultats de deux variantes de l'algorithme hiérarchique. La méthode standard, désignée sous le terme *hiérarchique* s'appuie sur l'utilisation de transferts point-à-point entre chaque hôte et sur l'emploi d'échanges en mémoire partagée au sein de chaque nœud. La variante non bloquante pipelinée (*hiérarchique/nb-pipelinée*) combine des opérations point-à-point non bloquantes pour les transferts intra et inter-nœuds et divise les messages en segments de 256 ko pour pipeliner les étapes de l'algorithme. Les deux variantes montrent bien que notre élection NUIOA des leaders apporte une amélioration de performance notable avec un gain allant de 10% à 25% selon l'algorithme utilisé.

De plus, notre élection des leaders pour l'opération de broadcast entre 96 cœurs sur le cluster Mirage dépasse nos attentes (Figure 5.19). Pour de larges messages, elle offre un gain de débit de 18% et 40% en fonction de l'algorithme hiérarchique, alors que nos expériences précédentes laissaient présager un impact moindre sur ces architectures. Nous pensons que ce large bénéfice est dû à la combinaison de l'amélioration des performances de transferts inter-nœuds et de la suppression des contentions supplémentaires sur le trafic mémoire évoquée en Section 5.2.1.

Nous ne cherchons pas ici à comparer les différentes implémentations d'opérations collectives mais à mettre en avant que l'amélioration de performance portée par notre stratégie d'élection NUIOA des leaders peut bénéficier à différents modèles et réglages de collectives hiérarchiques. De plus, nous avons pu observer que la prise en compte de la localité NUIOA peut également servir des implémentations d'opérations collectives non hiérarchiques mais structurées, telles que celle offerte par le module Tuned. Celle-ci est basée sur la propagation du message sur une chaîne linéaire de processus de façon pipelinée, avec une taille de segment dépendant de la largeur du message. Ce dernier est diffusé de proche en proche le long de cette chaîne de processus depuis la racine de l'opération. Nous avons réorganisé cette chaîne de sorte que le premier processus recevant les données sur chaque nœud soit à proximité de l'interface INFINIBAND de nos hôtes. Les résultats présentés Figure 5.20 montrent que le composant Tuned est peu adapté aux architectures du cluster Borderline, avec un débit n'excédant pas 130 Mo/s pour des messages inférieurs à 16 Mo [10]. Au delà de cette taille, il offre cependant un bien meilleur débit et profite de la réorganisation de la chaîne de processus intégrant la localité NUIOA.

10. Soit jusque 3 fois moins élevé que les performances du composant hiérarchique.

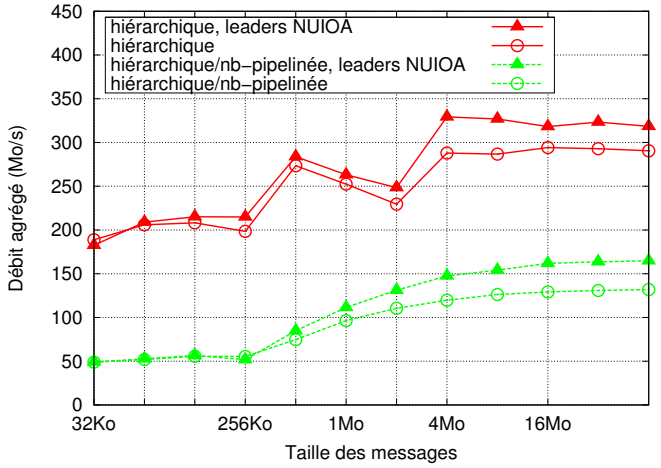


FIGURE 5.18 – *Performances du test IMB Bcast entre 64 processus (un par cœur, huit par nœud) sur la grappe Borderline pour différents algorithmes hiérarchiques de la bibliothèque* OPEN MPI.

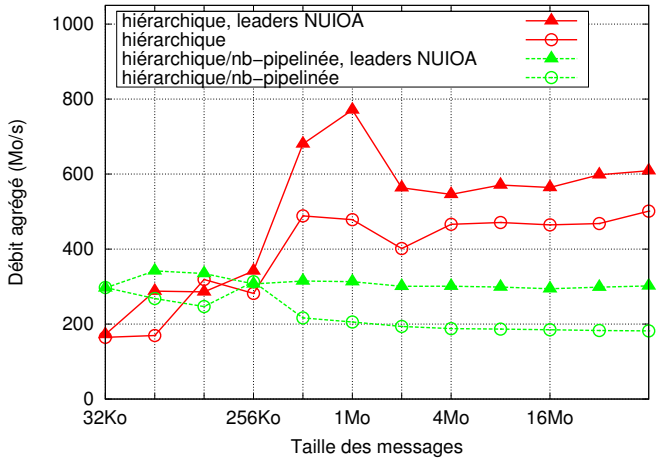


FIGURE 5.19 – *Performances du test IMB Bcast entre 96 processus (un par cœur, douze par nœud) sur la grappe Mirage pour différents algorithmes hiérarchiques de la bibliothèque* OPEN MPI.

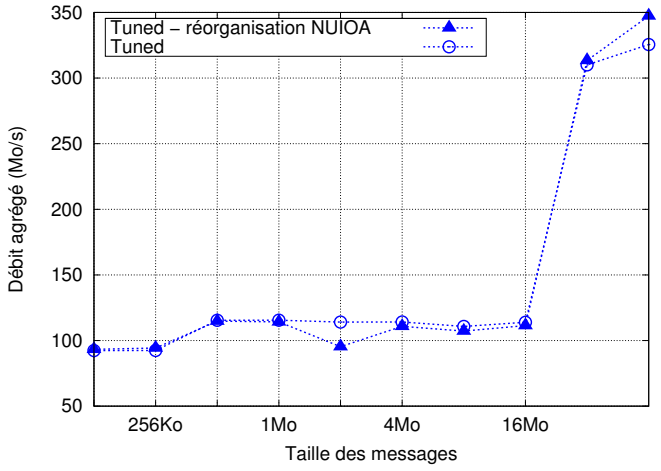


FIGURE 5.20 – *Performances du test IMB Bcast entre 64 processus (un par cœur, huit par nœud) sur la grappe Borderline pour le module Tuned, avec et sans réorganisation de la chaîne de processus.*

Passage à l'échelle de l'élection NUIOA des leaders

Nous nous intéressons ici aux effets de notre élection des leaders NUIOA en fonction du nombre de nœuds et de la charge de processus sur chacun. La Figure 5.21 présente les débits agrégés du test IMB Bcast pour un nombre croissant de processus sur huit nœuds de calcul de la plateforme Borderline. L'amélioration de performance de notre stratégie décroit de 50% à 10% alors que le nombre de transferts locaux augmente avec le nombre de processus par nœud, tandis que le nombre de communications inter-nœuds reste constant.

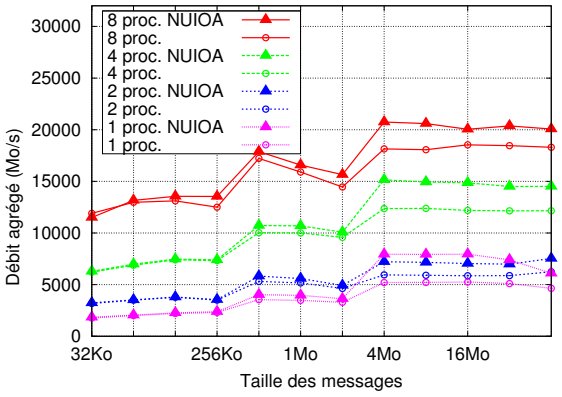


FIGURE 5.21 – *Débits agrégés pour un broadcast hiérarchique avec 1, 2, 4, ou 8 processus sur 8 nœuds de la grappe Borderline avec ou sans notre élection de leader* NUIOA.

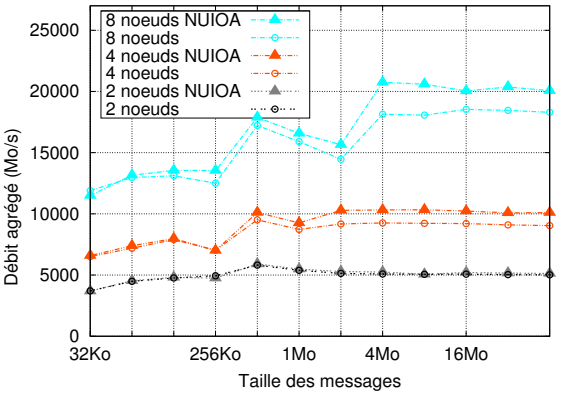


FIGURE 5.22 – *Débits agrégés pour un broadcast hiérarchique entre 2, 4 ou 8 nœuds (8 processus par nœud) utilisant ou non notre élection de leader* NUIOA.

Au contraire, la Figure 5.22 fait varier le nombre de nœuds utilisés en maintenant un processus par cœur sur chacun. L'impact de notre sélection NUIOA du leader croit bien avec l'augmentation du nombre de nœuds et donc de transferts réseau. Le bénéfice de notre stratégie passe ainsi de 2% pour deux nœuds jusqu'à 10% pour un total de huit nœuds.

Ces résultats montrent que notre proposition peut servir les performances sur les clusters modernes disposant de nombreux nœuds de calcul multicœurs. Elle améliore la partie inter-nœuds des opérations collectives sans en affecter la partie intra-nœud, et avantage ainsi les performances globales des opérations collectives du moment que le nombre de nœuds n'est pas radicalement inférieur à celui de cœurs par machine.

Une limitation potentielle de notre proposition concerne une éventuelle surcharge des leaders NUIOA. En effet, l'algorithme par défaut distribue la charge de leader sur l'ensemble des processus locaux en fonction de la racine de chaque opération. Notre stratégie ne distribue en revanche ce travail que sur les cœurs près de la carte réseau, ce qui peut augmenter leur charge relative. Cependant, comme les serveurs contemporains utilisent généralement 2 ou 4 sockets à 4 cœurs ou plus, concentrer cette charge de travail sur les cœurs d'un seul socket n'engendre pas un déséquilibre trop important. Quoi qu'il en soit, en attendant l'arrivée des opérations collectives non-bloquantes dans la révision 3.0 du standard MPI, les processus non-leaders ne peuvent pour l'instant pas faire progresser leurs calculs pendant que le leader fait progresser la collective. La surcharge du leader est donc de toute façon masquée par une attente bloquante de ses voisins.

5.2.3 Bilan

Les opérations collectives invoquées dans le cadre des applications MPI, se décomposent en communications intra-nœud effectuées en mémoire partagée entre les processus locaux et en transferts inter-nœuds effectués sur le réseau. Nous avons proposé de prendre en compte les aspects NUIOA des architectures contemporaines au sein des opérations collectives hiérarchiques pour améliorer l'efficacité des communications réseau.

Nos travaux préliminaires dans ce domaine reposent sur une élection des processus leaders responsables des transferts réseau, déterminée en fonction de la localité des processus et des cartes réseau. Nous avons implémenté notre stratégie au sein de l'opération de broadcast hiérarchique de l'implémentation OPEN MPI et noté que cette méthode peut effectivement améliorer les performances globales de cette opération. Nous observons ainsi un gain de débit allant jusqu'à 25% pour 64 processus répartis sur 8 nœuds du cluster Borderline, et jusqu'à 40% pour 96 processus distribués sur les hôtes Mirage. En contrebalançant la perte de débit occasionnée lorsque les processus leaders se trouvent à distance des cartes, il est ainsi possible d'améliorer notablement les performances. Nous avons également évalué que le bénéfice de cette stratégie est relatif au type d'opération collective invoquée (all-to-one, one-to-all et all-to-all) en fonction des architectures cibles, ainsi qu'au nombre de transferts réseau relatif au nombre d'échanges intra-nœud.

5.3 Vers des stratégies de transfert réseau adaptées à la localité des cartes

Alors que le nombre de cœurs par nœud de calcul augmente, le passage à l'échelle dans les serveurs contemporains est assuré par l'emploi de technologies d'interconnexion performantes qui ont non seulement réintroduit les architectures NUMA mais sont également souvent à l'origine d'accès non uniformes aux entrées-sorties (NUIOA). La localité des tâches de communication relative à la position physique des interfaces réseau a un impact notable sur les performances des communications réseau. Celle-ci devrait ainsi être prise en compte dans le placement des tâches de communications ou dans les stratégies mises en œuvre au cours des transferts.

Dans le cadre des applications MPI, le placement de l'ensemble des processus peut avoir été défini pour servir les affinités entre les processus, interdisant de revoir ce placement sans compromettre les besoins applicatifs. Nous nous sommes intéressés à la méthode inverse visant à prendre en compte les effets NUIOA au sein des stratégies de transfert des bibliothèques de communication. Dans ce contexte, nous avons dans un premier temps étudié la façon dont un processus donné devrait idéalement exploiter les cartes réseau et avons proposé d'adapter l'utilisation de multiples cartes réseau employées pour des transferts multirails. Il résulte de notre étude que les performances des communications réseau peuvent être significativement améliorées en ajustant la quantité de données envoyées sur chaque carte en fonction de la position physique de celles-ci et des processus communicants, et selon le schéma de communication. Nous avons également démontré qu'il est possible d'adapter le choix du processus qui devrait idéalement exploiter une carte réseau. Nous avons ainsi intégré une prise en compte de la localité NUIOA au sein des algorithmes hiérarchiques d'opérations collectives en sélectionnant un processus chargé des com-

munications réseau en fonction de la localité des cartes.

Les résultats probants de ces deux propositions témoignent de l'amélioration qui peut être extraite de la prise en compte de la topologie des architectures contemporaines sur les performances réseau dans les bibliothèques de communication. Ces travaux devraient toutefois être intégrés dans un contexte plus général en tenant compte de ce que font l'ensemble des processus et des cartes réseau simultanément pour harmoniser la sélection des processus et l'utilisation des cartes en accord avec les phénomènes de contention ou de surcharge qui peuvent apparaître dans un contexte applicatif.

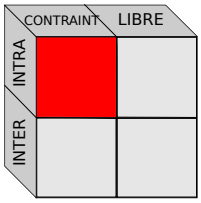
Si la topologie des architectures contemporaines influence les performances des transferts réseau, elle est également responsable d'une organisation hiérarchique des ressources mémoire à l'origine de temps d'accès variables aux données présentes sur les différents niveaux de cette hiérarchie. Les performances des échanges de données entre des tâches locales à un même calculateur sont ainsi directement liées au partage de ressources existant entre les tâches. Nous proposons ainsi d'étudier l'impact de la topologie mémoire sur les performances de transfert en mémoire partagée.

Chapitre 6

Adaptation des communications en mémoire partagée

Sommaire

6.1 Communications en mémoire partagée dans MPICH2 108
6.1.1 Transferts de larges messages dans MPICH2-NEMESIS 108
6.2 Analyse des performances 114
6.2.1 Plateforme de test et méthodologie 114
6.2.2 Communications point-à-point 116
6.2.3 Opérations collectives 124
6.2.4 NAS Parallel Benchmarks 129
6.3 Bilan 130



Le standard MPI est aujourd'hui favori dans l'exploitation des grappes de calcul tant pour les transferts réseau que pour les transferts intra-nœud. Alors que la multiplication des cœurs au sein des machines de calcul implique une augmentation du nombre de communications intra-nœud, les bibliothèques MPI contemporaines sont tenues d'assurer des transferts entre les processus locaux à un nœud de calcul, aussi efficaces que possible pour soutenir de bonnes performances applicatives. Les implémentations MPI disposent généralement de canaux de communication distincts pour les échanges réseau et les transferts locaux, qui peuvent ainsi bénéficier de l'utilisation de la mémoire partagée.

Dans le cadre d'une collaboration avec l'équipe Radix du laboratoire d'Argonne (*Argonne National Laboratory*) l'équipe Runtime a contribué à l'élaboration et l'étude de stratégies de communication en mémoire partagée au sein de l'implémentation portable MPICH2. Ces travaux ont notamment abouti à la création d'une stratégie de transfert direct avec l'assistance du noyau, basée sur l'utilisation du module noyau KNEM.

Les différentes méthodes de transfert intra-nœud supportées par MPICH2 sont détaillées dans la première partie de ce chapitre. Les cœurs localisés sur un même nœud de calcul partagent de la mémoire et une partie de la hiérarchie de caches susceptibles d'influencer les performances

exhibées par ces stratégies. La méthode optimale pour chaque échange est ainsi propre aux caractéristiques de celui-ci et aux spécificités matérielles intervenant entre les tâches communicantes en fonction de leur placement. Une mise en œuvre judicieuse de ces techniques de transfert consiste ainsi à une exploitation conjointe de celles-ci, orientée par les critères de communication et la topologie matérielle. Nous avons ainsi cherché à estimer l'influence du partage de ressources entre les cœurs sur chaque méthode de transfert, et proposons une étude approfondie de leurs performances sur différentes architectures matérielles.

6.1 Communications en mémoire partagée dans MPICH2

MPICH2 est une implémentation du standard MPI conçue pour offrir un bon niveau de performance sur un très large éventail d'architectures, et pour de multiples technologies réseau. Elle s'inscrit dans le cadre des supports exécutifs portables destinés au calcul haute performance et fait partie des implémentations MPI les plus populaires. Cette bibliothèque s'appuie sur un sous-système de communication nommé NEMESIS [63] qui effectue les transferts inter-nœuds au travers des interfaces de réseaux rapides disponibles et les échanges intra-nœud en mémoire partagée. Pour les transferts intra-nœud auxquels nous nous intéressons dans ce chapitre, NEMESIS offre une gestion séparée des petits messages nécessitant une faible latence, et des larges messages pour lesquels une bande passante importante est primordiale. L'optimisation des communications par petits messages ayant déjà été largement étudiée [64], nos travaux se concentrent sur les communications intra-nœud par larges messages (au moins de l'ordre de la dizaine de kilo-octets).

6.1.1 Transferts de larges messages dans MPICH2-NEMESIS

L'une des améliorations apportée par le sous-système de communication NEMESIS réside dans l'introduction de l'interface *Large Message Transfer* (LMT), qui a été conçue de façon suffisamment générale pour supporter plusieurs mécanismes de transfert de larges messages. Elle est notamment capable de supporter des communications unilatérales (*one-sided*) et bilatérales (*two-sided*). Cette flexibilité permet ainsi à NEMESIS d'employer le mécanisme le plus adéquat disponible, sur chaque plateforme et pour chaque transfert spécifique. La Figure 6.1 illustre une partie de la pile logicielle de MPICH2. Au delà d'une taille de message suffisamment importante (fixée à 64 ko), le LMT prend en charge les transferts. Il peut ainsi sélectionner le mécanisme le plus intéressant, utilisant des tampons en mémoire partagée (*shm-copy*), une copie au travers d'un tube UNIX (*vmsplice*) ou encore une copie directe via un module noyau (*knem*).

Copie double-buffering

Le premier mécanisme de transfert de larges messages mis au point dans MPICH2-NEMESIS repose sur l'utilisation de tampons (*buffers*) alloués dans une zone de mémoire partagée entre les processus, qui tiennent le rôle d'intermédiaires de communication. Un processus émetteur copie ses données dans un tampon partagé depuis lequel un processus récepteur les copie à son tour vers la zone mémoire destination du message comme illustré en Figure 6.2 (a). Ce procédé requiert

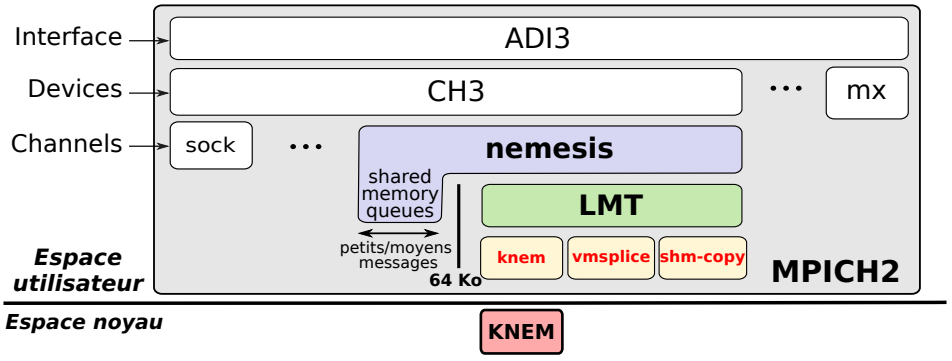
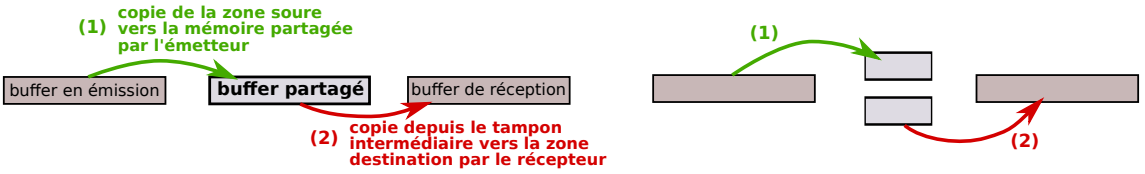


FIGURE 6.1 – *Représentation partielle de la pile logicielle de* MPICH2. *Le transfert de larges messages pris en charge par le sous-module de communication* NEMESIS *repose sur l'utilisation de l'interface Large Message Transfer qui supporte diverses stratégies de transfert intra-nœud (shm-copy, vmsplice, knem).*

ainsi deux copies et nécessite une synchronisation pour assurer que la lecture dans le tampon soit faite après la fin de l'écriture correspondante.

Pour les messages de grande taille cette approche simpliste offre des performances limitées. L'allocation d'un tampon dédié à chaque message de grande taille peut tout d'abord avoir un impact négatif sur la mémoire disponible. Elle implique également une forte latence inhérente au fait que le récepteur doive attendre que la totalité des données ait été transférée dans le buffer partagé, avant de commencer la recopie depuis celui-ci. Pour limiter ces inconvénients, la méthode employée utilise une paire de tampons plus petits permettant un recouvrement partiel des copies. Lorsqu'un processus lit des données depuis le premier tampon, l'autre écrit des données dans le second, puis inversement (Figure 6.2 (b)). Cette approche nommée *double-buffering* [1] réduit la latence et améliore la bande passante grâce à l'exécution des copies en parallèle par les deux processus mis en jeu. Les performances globales dépendent de la taille des buffers utilisés : si trop petits, le débit souffre des synchronisations fréquentes et ne profite pas de la bande passante mémoire ; si trop grands, les intérêts du *double-buffering* ne sont pas exploités pour les messages de taille moins importante. MPICH2 utilise des tampons de 64 ko, bon compromis entre latence et débit sur de nombreuses architectures.



(a) Tansfert inter-processus utilisant un tampon en mémoire partagée *(b) Tansfert double-buffering*

FIGURE 6.2 – *Transferts de données en mémoire partagée utilisant :*
(a) un seul tampon intermédiaire, les deux copies doivent être faites l'une après l'autre.
(b) une paire de tampons intermédiaires et permettant un recouvrement partiel des copies.

1. Disponible dans la pile logicielle de MPICH2 dans le module *shm-copy*.

Cette méthode de communication bilatérales est très largement portable (disponible sur l'ensemble des systèmes dotés de l'appel système `mmap` ou de mémoire partagée System V). Elle utilise cependant deux processeurs, les rendant de ce fait peu disponibles pour des phases de calcul, et entraîne une forte pollution des caches en y remplaçant des données utiles à l'application par les données échangées.

Copie directe au travers d'un tube appuyée sur l'appel système `vmsplice`
Pour éviter de solliciter deux processus tout au long du transfert et réduire ainsi la consommation processeur, une solution est d'effectuer le transfert à l'aide d'une seule copie. Dans les environnements UNIX tels que LINUX, un processus ne peut pas accéder directement à l'espace d'adressage d'un autre processus. Pour pouvoir mettre en place une copie directe, il faut donc solliciter le système d'exploitation. Dans le cadre de la collaboration des équipes Radix et Runtime, une méthode implémentée dans MPICH2-NEMESIS repose sur l'utilisation associée d'un tube de communication UNIX (*pipe*) et de l'appel système `vmsplice` [16] introduit dans le noyau LINUX 2.6.17.

Un transfert classique au travers d'un tube UNIX nécessite une copie à l'entrée du tube, effectuée par l'appel système `write`, puis une copie depuis la sortie accomplie par l'appel à `read` (Figure 6.3). L'utilisation de l'appel système `vmsplice` permet d'éviter la copie en émission. Les processus émetteur et récepteur ouvrent un tube dans lequel le processus émetteur va *attacher* des pages mémoire grâce à `vmsplice`. Lorsque le processus récepteur utilise l'appel système `read`, les données sont alors directement copiées depuis ces pages vers la zone mémoire destination (Figure 6.4).

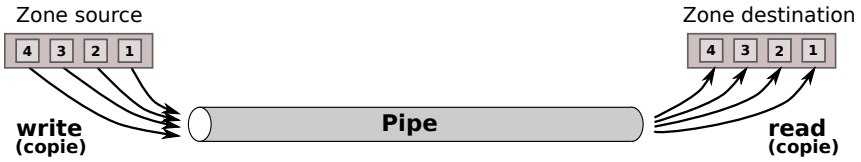


FIGURE 6.3 – *Transfert classique au travers d'un tube UNIX. Une copie de chaque page est nécessaire en émission (appel à* `write`*) et en réception (appel à* `read`*).*

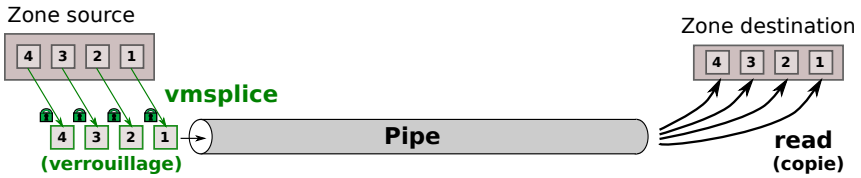


FIGURE 6.4 – *Transfert au travers d'un tube appuyé sur l'appel système* `vmsplice` *permettant une seule copie par page. Les pages sont verrouillées en émission où elles ne nécessitent pas d'être recopiées. L'appel à* `read` *en réception engendre une copie de chaque page vers la zone destination.*

Par défaut, le noyau Linux limite le nombre de pages maximum par tube à 16 (4 ko/page), et donc à un maximum de 64 ko transférés par appel à `vmsplice` et `read`. A première vue embarrassante, cette limitation profite toutefois à la réactivité de NEMESIS en permettant périodiquement la scrutation de nouveaux messages entre les segments envoyés. Sur les processeurs modernes, les multiples appels à `vmsplice` pour un même transfert génèrent chacun un surcoût de l'ordre d'une centaine de nanosecondes. En considérant le temps de copie d'un tronçon de 64 ko (environ 8μs pour une bande passante mémoire de 8 Go/s), ce coût semble acceptable pour maintenir la réactivité.

Le LMT `vmsplice` offre ainsi un support de copie directe générique qui évite les recopies supplémentaires induites par le double-buffering. Il a l'avantage de fonctionner sur tous les systèmes LINUX récents, mais supporte uniquement une interface synchrone et bloquante.

Copie directe au travers du module noyau KNEM

Dans le cadre de la collaboration entre l'équipe Runtime et l'équipe Radix de l'ANL, la mise en commun de nos travaux a conduit au développement d'une stratégie de copie directe basée sur un module noyau dédié. L'objectif est d'outrepasser les limitations du LMT vmsplice, et d'offrir de meilleures performances grâce à un modèle adapté aux communications MPI.

Cette idée a précédemment été étudiée dans l'implémentation MVAPICH dont le module noyau LIMIC2 [82] offre une implémentation multisupport. Le module LINUX KNEM (*Kernel* NEMESIS) proposé va plus loin en supportant notamment les communications non-bloquantes, asynchrones et vectorielles [2] et en profitant des fonctionnalités de déport de copie mémoire dans le matériel. Il s'apparente également aux travaux menés par Vaidyanathan *et al.* exploitant le déport de copie offert par la technologie I/OAT des contrôleurs INTEL [150, 149].

Le module KNEM a été généralisé à partir de la gestion intra-nœud d'OPEN-MX [3] [115, 116]. Il s'appuie sur un périphérique caractère (`/dev/knem`) qui implémente deux principales commandes de communication (Figure 6.5). Le LMT du processus émetteur (1) déclare une zone mémoire d'émission à KNEM et (2) obtient un *cookie* en retour (commande *Send*). Ce *cookie* est ensuite envoyé au récepteur au travers de l'habituel message de *rendez-vous* MPI (3). Le LMT récepteur qui souhaite recevoir les données dans une zone mémoire déclare cette zone à KNEM (4) en lui associant le *cookie* d'émission (commande *Recv*). Après avoir récupéré la zone mémoire d'émission via le *cookie* (5), le module KNEM prend en charge le transfert de données (6) d'une zone mémoire à l'autre directement depuis le noyau LINUX.

La technologie I/OAT (*Input/Output Acceleration Technology*) implémentée dans les contrôleurs mémoire INTEL [147] comprend un dispositif matériel, le moteur DMA (*DMA Engine*), capable d'effectuer efficacement des copies mémoire en arrière plan. Il libère ainsi le processeur de ce transfert au profit de travail "utile". Ce modèle empêche par la même occasion la pollution des caches, exonérés des données relatives à la copie. Pour profiter de ces avantages, la possibilité d'utiliser le déport de copie I/OAT a été ajoutée au module KNEM. Lorsque le moteur DMA est disponible, cette option peut être activée par le LMT KNEM via un paramètre supplémentaire à la commande de réception.

2. Permettant le transfert de données vers ou depuis des zones mémoire non-contigues.
3. OPEN-MX exporte le potentiel de la pile logicielle MX sur des réseaux Ethernet.

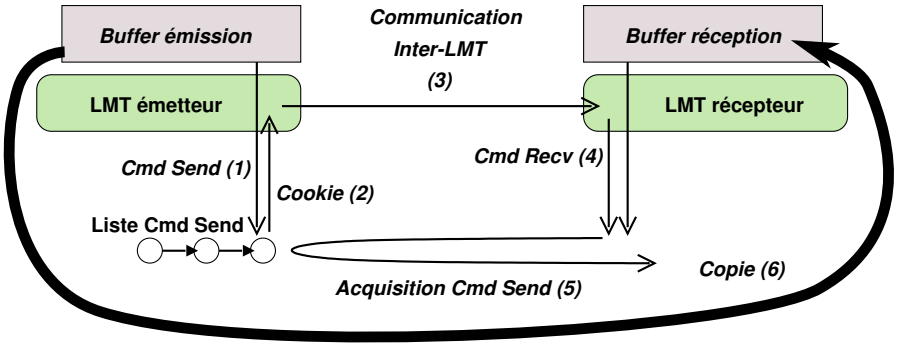


FIGURE 6.5 – *Transfert de larges messages avec le module noyau KNEM.*

Un des problèmes techniques induits par ce modèle est lié au verrouillage des zones mémoire mises en jeu. En effet, il n'y a aucune garantie qu'une adresse virtuelle (utilisée par les applications) sera toujours projetée au même endroit en mémoire physique (dont les adresses sont utilisées par le matériel et donc par I/OAT). La gestion de la mémoire virtuelle par le système d'exploitation pourrait par exemple, en cas de déficit mémoire, évincer des pages sur le disque (*swap*). Si cela se produisait pendant une copie I/OAT, les données manipulées deviendraient invalides. Pour assurer que la projection mémoire ne change pas pendant le transfert, le module KNEM punaise les zones mémoire de l'application en mémoire physique avant l'intervention du moteur DMA. Cette précaution est d'ailleurs appliquée en l'absence d'I/OAT, puisque le processus récepteur ne peut pas facilement accéder à l'espace d'adressage de l'émetteur sous LINUX. Ces opérations de verrouillage mémoire sont intégrées aux commandes d'émission et réception du pilote KNEM. Le LMT de MPICH2 n'a donc qu'à utiliser ces commandes sans se soucier de ces problèmes de verrouillage sous-jacents.

Alors que le moteur DMA matériel libère le processeur de l'hôte et effectue les copies en arrière plan, il permet un potentiel recouvrement des copies par du calcul. De plus, l'implémentation de NEMESIS en espace utilisateur induit la capacité de scruter périodiquement l'arrivée de nouveaux messages. Il est ainsi possible d'avoir une implémentation de KNEM asynchrone laissant le processus récepteur repasser en espace utilisateur tandis que la copie est effectuée en arrière plan. En pratique, cette implémentation nécessite d'enlever l'attente de terminaison bloquante dans KNEM. Pour assurer la notification, la bibliothèque passe au module noyau l'adresse d'une variable de statut. A la fin du transfert, le module noyau note le statut à `Success`. La bibliothèque peut ainsi scruter cette variable pour vérifier la terminaison. Ce modèle est assez compliqué à implémenter car l'interface matérielle d'I/OAT est elle-même basée sur une scrutation. Puisque qu'aucune interruption n'intervient dans I/OAT, KNEM doit ainsi scruter la terminaison I/OAT pour savoir quand mettre à jour la variable de statut (Figure 6.6).

Plutôt que de gaspiller du temps processeur pour cette vérification, le modèle asynchrone bénéficie du fait que les requêtes de copie I/OAT sont traitées dans l'ordre de leur soumission. Après la terminaison de la copie requise pour la communication MPI, KNEM soumet à I/OAT une petite requête de copie permettant de mettre à jour la valeur de la variable de statut (Figure 6.7). De cette façon, le transfert des données comme la terminaison sont menés en arrière plan permettant

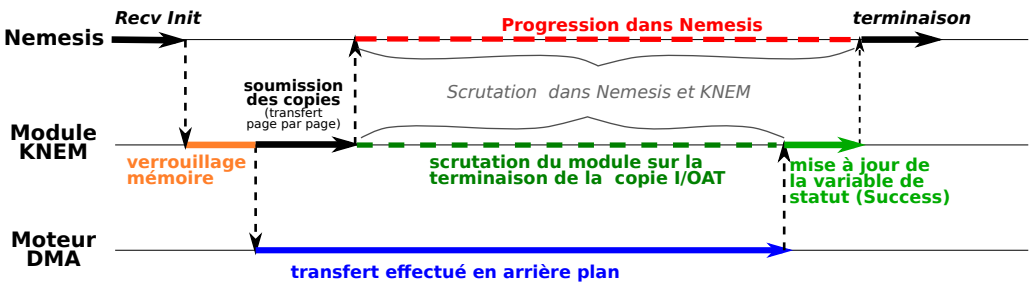


FIGURE 6.6 – *Détection de la terminaison d'une copie asynchrone effectuée en arrière plan par la technologie* I/OAT. *Dans ce modèle une scrutation est faite par le module noyau* KNEM *pour détecter la fin de la copie par le moteur DMA.* KNEM *met alors à jour la variable de statut elle-même scrutée par* NEMESIS.

le recouvrement par le calcul. Les performances de transfert bénéficient de ce modèle [4] puisque le travail est effectué en tâche de fond. Par défaut, lorsque KNEM est activé avec I/OAT, le mode utilisé est ainsi basculé sur le mode asynchrone.

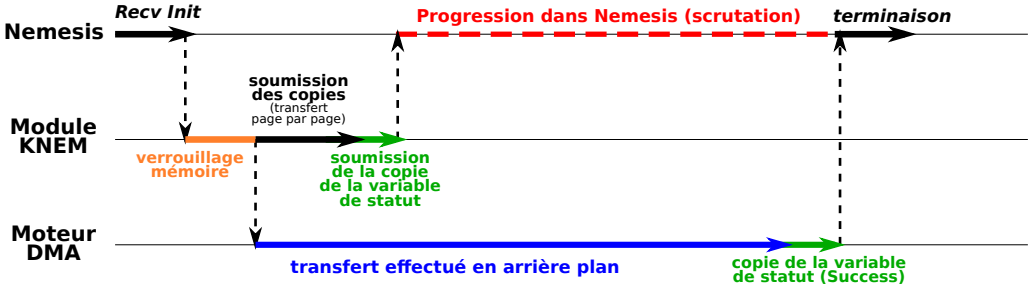


FIGURE 6.7 – *Détection de la terminaison d'une copie asynchrone effectuée en arrière plan par la technologie* I/OAT. *La variable de statut est directement mise à jour par une seconde copie* I/OAT. *Le module* KNEM *est ainsi libéré de la détection de la fin du transfert* I/OAT.

Un modèle asynchrone a également été implémenté pour les transferts directs sans I/OAT en confiant la copie à un thread noyau exécuté sur le cœur du processus récepteur. Ce modèle permet le recouvrement des copies mémoire mais implique en contrepartie une compétition entre le processus NEMESIS en espace utilisateur et le thread noyau assigné à la copie. Cette stratégie réduit ainsi les performances de transfert[4], mais autorise une copie asynchrone avec une progression du calcul en espace utilisateur.

Le LMT KNEM mis en place permet ainsi d'effectuer des copies directes adaptées au transfert de données MPI pour lequel il a été spécifiquement conçu. Il peut également bénéficier du déport de copie I/OAT disponible sur les contrôleurs INTEL et dispose de modes de transfert synchrone et asynchrone. Son utilisation, à priori préférable à celle du LMT vmsplice, est cependant restreinte aux plateformes de calcul sur lesquelles le chargement d'un module noyau personnalisé est acceptable.

4. Une comparaison des performances de KNEM en mode synchrone/asynchrone est disponible en Annexe B.1. A l'exception du cas où le déport de copie I/OAT est employé, les performances présentées dans la suite de ce Chapitre résultent de l'utilisation du mode synchrone de KNEM, plus performant.

Bilan : différentes stratégies de transfert
Le sous-module de communication NEMESIS dispose ainsi de plusieurs stratégies de transfert de larges messages exploitables au sein de l'interface LMT. En réduisant le nombre de copies nécessaires au transfert de larges messages, la stratégie *vmsplice*, disponible sur les noyaux Linux récents, devrait normalement offrir une amélioration de performance comparée à la stratégie *double-buffering*. Au contraire de cette méthode générique, le module noyau KNEM a été spécifiquement conçu pour effectuer des transferts de données MPI. Il devrait ainsi surpasser le LMT vmsplice en terme de performances, et son utilisation devrait être favorisée lorsque l'emploi d'un module noyau personnalisé est possible. L'exploitation du LMT KNEM soulève également la question de savoir quand utiliser le déport de copie I/OAT.

Les mécanismes de transfert en mémoire partagée sont connus pour présenter des performances variables [142]. Nous proposons une évaluation des stratégies employées par l'interface LMT de MPICH2-NEMESIS sur de multiples architectures, pour dégager l'impact de la topologie matérielle sur les performances. L'objectif sous-jacent est de pouvoir extraire de cette analyse un choix de la stratégie à utiliser par le LMT en fonction des caractéristiques matérielles et de transfert.

6.2 Analyse des performances

Nous présentons dans cette section une évaluation des performances des LMTs de MPICH2-NEMESIS et tout particulièrement du LMT KNEM au niveau des communications point-à-point et dans le cadre d'opérations collectives et d'applications MPI sur diverses architectures.

6.2.1 Plateforme de test et méthodologie

Notre plateforme de test est composée de plusieurs architectures INTEL de topologies variées, dont les spécificités sont résumées en Table 6.1. Pour plus de clarté les différents processeurs assemblés au sein de ces machines sont illustrés en Figures 6.8, 6.9 et 6.10. Une description globale est fournie en Annexe A. Sur ces architectures, de nombreux placements sont possibles. Deux processus peuvent être exécutés sur des cœurs :

- qui partagent un cache de niveau L2 et/ou L3 ;
- au sein d'un même socket, sans cache commun ;
- appartenant à des sockets différents au sein du même nœud NUMA ;
- ou appartenant à des nœuds NUMA distincts.

Notre analyse des performances se décompose en 3 axes :

- l'examen des transferts point-à-point, effectués entre deux processus MPI ;
- l'observation des opérations collectives, intervenant entre les différents processus d'un groupe et pouvant être divisées en trois types, *one-to-all*, *all-to-one* et *all-to-all* ;
- et enfin l'étude de performances applicatives.

Machine	*Architecture*	*Caches partagés*	*Cœurs*	I/OAT
Bill	**2 processeurs** quadri-cœurs XEON Clovertown E5345 (2,33 GHz)	**L2** (4 Mo) partagés entre paire de cœurs (Fig. 6.8)	**8** cœurs	oui
Hannibal	**2 processeurs** quadri-cœurs XEON Nehalem X5550 (2,66 GHz) 2 nœuds NUMA, hyperthreading ignoré	**L3** (8 Mo) partagé entre les 4 cœurs d'un socket (Fig. 6.9)	**8** cœurs	oui
Idkonn	**4 processeurs** hexa-cœurs XEON Dunnington X7460 (2,66 GHz)	**L3** (16 Mo) partagé entre les 6 cœurs d'un socket **L2** (3 Mo) partagés entre paire de cœurs (Fig. 6.10)	**24** cœurs	oui
Bertha	**4 nœuds NUMA** chacun équivalent à Idkonn ($4*4$ hexa-cœurs)	**L3** (16 Mo) partagé entre les 6 cœurs d'un socket **L2** (3 Mo) partagés entre paire de cœurs (Fig. 6.10)	**96** cœurs	non disponible

TABLE 6.1 – *Caractéristiques des machines de notre plateforme de test.*

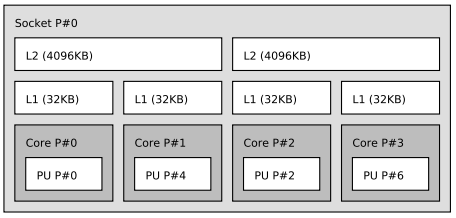


FIGURE 6.8 – *Représentation d'un socket de la machine bi-processeur (Clovertown) Bill, obtenue d'après la bibliothèque* `hwloc`.

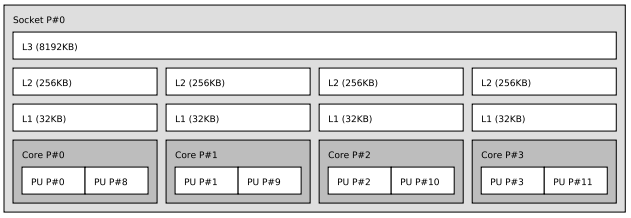


FIGURE 6.9 – *Représentation d'un socket des processeurs Nehalem de la machine Hannibal, obtenue d'après la bibliothèque* `hwloc`.

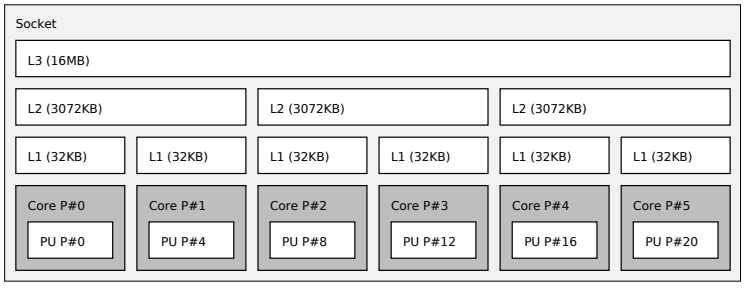


FIGURE 6.10 – *Représentation proposée par la bibliothèque* `hwloc` *pour un des 4 sockets d'un processeur Dunnington (utilisé dans les machines Idkonn et Bertha).*

La plupart de nos tests sont basés sur l'utilisation de la suite INTEL MPI *Benchmarks* [31] (IMB), qui offre des supports de transfert point-à-point (test *Pingpong*) et d'opérations collectives (*Allgather, Alltoall, Allreduce, Reduce, Scatter, Bcast*, etc.). La version 3.2 d'IMB dispose d'une option *offcache* qui empêche l'utilisation répétée des données contenues dans le cache. Pour clarifier l'effet d'une telle réutilisation, les Figures 6.11 et 6.12 présentent les résultats d'un ping-pong entre deux processus (avec et sans cache partagé) sur la machine Bill, effectué avec la stratégie NEMESIS double-buffering, et en fonction de l'activation de l'option *offcache*.

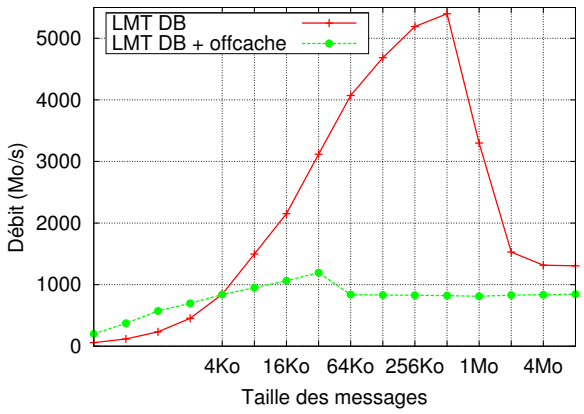


FIGURE 6.11 – *Débits de ping-pong IMB entre deux processus partageant un cache sur la machine Bill,* ***avec et sans l'option offcache*** *(stratégie double-buffering (DB)).*

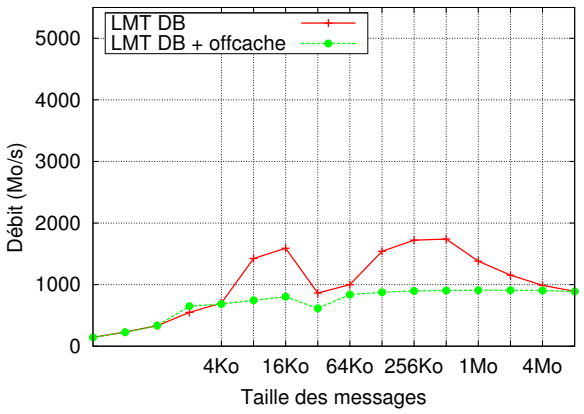


FIGURE 6.12 – *Débit d'un ping-pong IMB entre deux processus sur des puces différentes de la machine Bill,* ***avec et sans l'option offcache*** *(stratégie double-buffering (DB)).*

Les applications réelles utilisent généralement les caches pour les calculs, limitant la mise en cache possible des buffers de communication, et peuvent présenter une réutilisation "mitigée" des données du cache. Dans le cas des transferts point-à-point, nous présenterons nos observations avec et sans l'utilisation de cette option. Pour les opérations collectives qui génèrent généralement une plus grande concurrence sur les caches, l'utilisation de cette option devrait représenter des performances plus proches de celles attendues dans le cadre d'applications réelles. Cette option est donc activée pour les expériences proposées.

Les performances applicatives sont pour leur part évaluées grâce aux *NAS Parallel Benchmarks* [29], et plus précisément grâce aux tests FT et IS, caractérisés par un nombre important de transferts de larges messages. Pour satisfaire les conditions nécessaires à l'exécution de ces tests, un nombre de processus égal à une puissance de deux est utilisé. Sur les architectures Bertha et Idkonn dont le nombre de cœurs n'est pas une puissance de deux, une paire de cœurs adjacents est ignorée sur chaque socket, utilisé comme s'il s'agissait d'une puce quadri-cœur.

6.2.2 Communications point-à-point

Comparaison des stratégies de transfert

Les Figures 6.13 et 6.14 présentent le débit du test IMB *Pingpong* sur MPICH2 avec les différents LMTs (activés dès 64 ko) sur la machine Bill, sans l'option *offcache*.

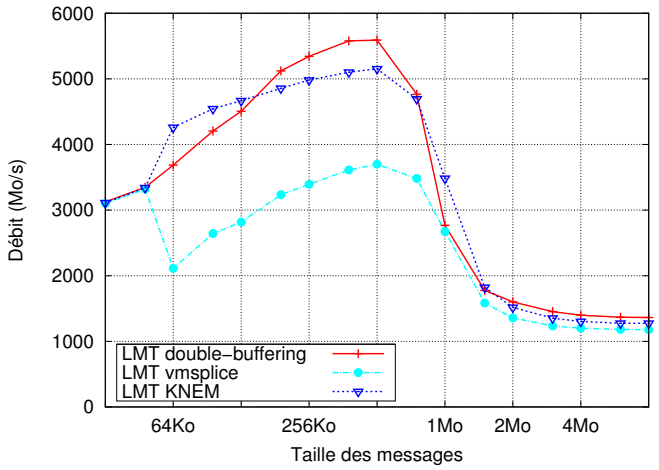


FIGURE 6.13 – *Pingpong IMB entre 2 processus partageant un cache L2 de 4 Mo (machine Bill).*

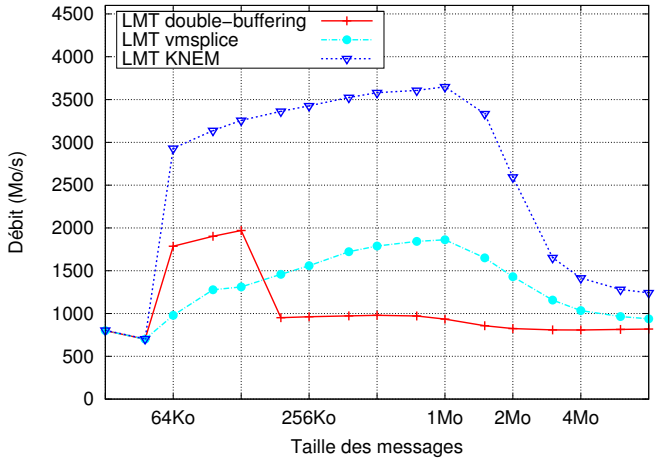


FIGURE 6.14 – *Pingpong IMB entre 2 processus ne partageant aucun cache (machine Bill).*

Le premier résultat intéressant est ici que les performances de la stratégie double-buffering dépendent bien plus du placement des processus que celles des stratégies de copie directe. En effet, en présence d'un cache partagé entre les processus communicants, cette stratégie présente d'excellentes performances atteignant 5,5 Go/s (Figure 6.13). Au contraire, lorsqu'aucun cache n'est partagé (Figure 6.14), les performances sont très largement réduites et la stratégie apparaît pauvre en comparaison avec les autres méthodes. Ce comportement s'explique simplement par le fait que les copies doubles invoquées utilisent d'avantage le cache que les transferts par copie directe, et sont ainsi fortement tributaires du partage de cache entre les processus.

La réduction du nombre de copies induite par le LMT vmsplice est ainsi plus intéressante que le *double-buffering* habituel lorsqu'aucun cache n'est partagé entre les cœurs sur lesquels s'exécutent les processus émetteur et récepteur. Le débit obtenu peut être jusqu'à deux fois supérieur. Si les cœurs partagent un cache, le surcoût de la copie supplémentaire est inférieur au gain obtenu grâce à la faible latence d'accès aux données contenues dans le cache, la stratégie de double-buffering reste alors plus efficace. A première vue, un compromis permettant d'activer `vmsplice` dynamiquement lorsqu'aucun cache n'est partagé semblerait adapté. Toutefois, les performances du LMT vmsplice apparaissent, comme attendu, de très loin inférieures à celles obtenues avec notre module noyau dédié KNEM. La stratégie vmplice s'inscrit donc comme une solution de secours dont l'intérêt

n'existe que dans le seul cas où l'utilisation de KNEM est proscrite [5]. Nous ne développerons ainsi pas davantage notre analyse des performances de ce LMT et nous focaliserons dans la suite de cette section sur l'utilisation des stratégies double-buffering et KNEM.

Le LMT KNEM, conçu pour des schémas d'accès spécifiques à MPI, améliore significativement les performances. Si aucun cache n'est partagé entre les cœurs, la stratégie KNEM classique [6] est plus de trois fois plus rapide que la copie double-buffering, atteignant 3,5 Go/s. Si les deux cœurs partagent un cache, les performances de KNEM s'avèrent sensiblement les mêmes que celles du double-buffering. Les observations faîtes sur des tests similaires sur nos différentes plateformes (résumées en Table 6.2) tendent à confirmer ces résultats. Avec la réutilisation des données contenues dans le cache (*offcache* désactivée), la copie directe au travers du module noyau KNEM améliore largement les performances de transfert comparée à la stratégie double-buffering lorsqu'aucun cache n'est partagé. Dans le cas contraire, les deux méthodes exhibent des performances relativement similaires sur nos machines 8 cœurs. Sur les machines Bertha et Idkonn, dotées de puces identiques avec des caches partagés de niveaux L2 et L3, la stratégie double-buffering s'avère plus efficace que KNEM dans les seuls cas ou les deux niveaux de caches sont partagés entre les cœurs sur lesquels s'exécutent les processus communicants. Sur ces processeurs (*Dunnington* XEON 74xx), le constructeur INTEL a rajouté a posteriori un niveau de cache L3 à des processeurs initialement conçus sans L3 (*Harpertown* XEON 54xx). Cette modification significative de l'architecture mémoire tient lieu d'ajustement temporaire spécifique à ce processeur [7] et pourrait justifier la sensibilité de la stratégie double-buffering observée ici.

L'activation de l'option *offcache* réduit les bandes passantes obtenues sur l'ensemble des machines mais amène à des constations relativement équivalentes. La Figure 6.15 montre le débit d'un ping-pong sur l'architecture Bertha [8] (Figure A.13) pour différents placements des processus : avec un partage de cache L2 et/ou L3, sur différentes sockets du même nœud NUMA, ou sur des nœuds NUMA distincts.

Comme attendu, l'usage important du cache par la stratégie double-buffering rend celle-ci plus sensible au placement des processus (avec un facteur 6) que KNEM (20%). Ses performances sont ainsi supérieures à celles de KNEM (d'environ 50%) dès lors qu'un cache est partagé et quelque soit la taille du message échangé. Dans le cas contraire, la réduction du nombre de copies de la stratégie KNEM permet d'obtenir une bande passante jusqu'à 2 fois plus élevée que celle exhibée par le double-buffering. Nous avons observé des comportements similaires sur les architectures Idkonn, Bill et Hannibal avec un impact minoré par la relative simplicité de leur topologie en comparaison de celle de Bertha et un bénéfice réduit du double-buffering en présence de caches partagés (Table 6.2).

5. Cette conclusion a été validée sur les différentes architectures de notre plateforme, y compris avec l'activation de offcache, pour les opérations collectives et pour des performances applicatives. Pour en témoigner, un certain nombre d'exemples complémentaires (disponibles en Annexe B.2 et B.4) incluent les performances de la stratégie vmsplice.

6. Par opposition à la méthode profitant du déport de copie I/OAT, notée par la suite KNEM +I/OAT.

7. Elle ne devrait pas apparaître sur les processeurs standard de la gamme.

8. I/OAT non supporté.

	option *offcache* désactivée	
Machine	*Avec caches partagés* *	*Sans cache partagé*
Bill (*) L2 (4 Mo)	KNEM et DB comparables (avantage léger DB) Bénéfice d'I/OAT après 1 Mo	Stratégie préférable : **KNEM** (gain entre 33% et 622%) Bénéfice d'I/OAT après 2 Mo
Idkonn (*) L3 (16 Mo) ou L2+L3 (3+16 Mo)	Stratégie préférable : - **KNEM** si L3 partagé (3% à 56%) - **DB** si L2+L3 partagés (0% à 22%) Bénéfice d'I/OAT après 4 Mo	Stratégie préférable : **KNEM** (gain de 38% à 79%) Bénéfice d'I/OAT après 8 Mo
Bertha (*) L3 (16 Mo) ou L2+L3 (3+16 Mo)	Stratégie préférable : - **KNEM** si L3 partagé (-11% à 55%) - **DB** si L2+L3 partagés (-9% à 25%)	Stratégie préférable : **KNEM** (gain de 58% à 80%)
Hannibal (*) L3 (8 Mo)	Stratégie préférable : **KNEM** (gain de 5% à 29% après 256 ko) I/OAT faible, pas de bénéfice	Stratégie préférable : **KNEM** (gain de 10% à 50% après 128 ko) I/OAT faible, pas de bénéfice

	option ***offcache*** activée	
Machine	*Avec caches partagés* *	*Sans cache partagé*
Bill (*) L2 (4 Mo)	KNEM et DB comparables (sans I/OAT, avantage léger DB) Stratégie idéale : **KNEM+I/OAT** bénéfice immédiat [9] (> +30%)	Stratégie préférable : **KNEM** (gain de 20% à 36%) Stratégie idéale : **KNEM+I/OAT** bénéfice immédiat [7] (+45 à 62%)
Idkonn (*) L3 (16 Mo) ou L2+L3 (3+16 Mo)	Stratégie préférable : - indifférente si L2+L3 partagés - **KNEM** si L3 partagé (+5/6%) Stratégie idéale : **KNEM+I/OAT** bénéfice immédiat [7] (+16% à 22%)	Stratégie préférable : **KNEM** (gain de 5% à 53%) Stratégie idéale : **KNEM+I/OAT** bénéfice immédiat [7] (+18% à 60%)
Bertha (*) L3 (16 Mo) ou L2+L3 (3+16 Mo)	Stratégie préférable : **DB** (après 256 ko, gain supérieur à 27% si L2+L3 partagés, à 20% si L3 partagé. Négligeable avant 256 ko)	Stratégie préférable : **KNEM** (gain de 32% à 62% si nœuds distincts, 9% à 50% sinon)
Hannibal (*) L3 (8 Mo)	KNEM et DB comparables (avantage léger KNEM) I/OAT faible, pas de bénéfice	Stratégie préférable : **KNEM** (gain de -6% à 19%) I/OAT faible, pas de bénéfice

TABLE 6.2 – *Résumé de la comparaison des LMTs double-buffering (DB) et* KNEM *activés dès 64 ko pour des communications point-à-point sur les différentes architectures de notre plateforme de test.*

9. Gain par rapport au double-buffering pour des messages entre 64 ko et 8 Mo.

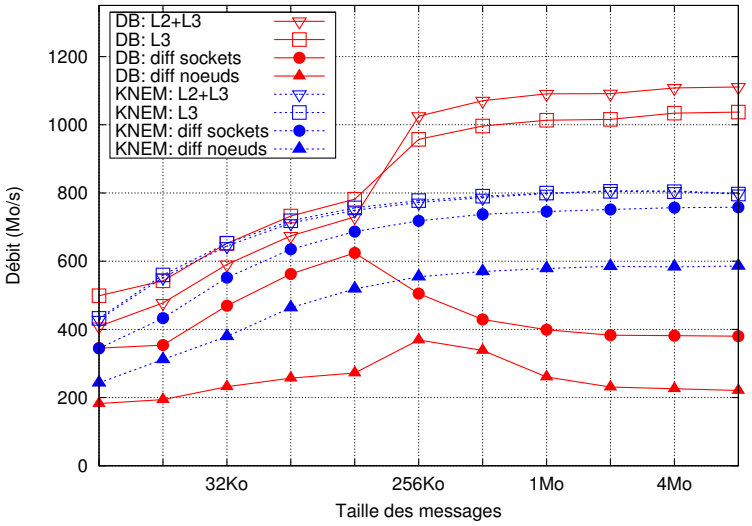


FIGURE 6.15 – *Débits du test IMB Pingpong (offcache activé) pour les LMTs double-buffering (DB) et* KNEM *en fonction du placement des processus sur la machine Bertha, sur des cœurs :*
- partageant des caches de niveau L2 (3 Mo) et L3 (16 Mo) : ***L2+L3***
- partageant un cache L3 (16 Mo) : ***L3***
- appartenant à des sockets différents : ***diff sockets***
- appartenant à des nœuds NUMA *distincts :* ***diff nœuds***

Bénéfices du déport de copie I/OAT

Les Figures 6.17 et 6.16 reprennent les performances obtenues sur l'architecture Bill pour la stratégie double-buffering et KNEM, avec et sans l'utilisation du déport de copie I/OAT. Elle témoignent de l'avantage du déport de copie par KNEM pour de très larges messages (supérieurs à 1 Mo avec un cache partagé (4 Mo), ou à 2 Mo dans le cas contraire) sur la machine Bill. La soumission des copies à I/OAT nécessite un accès au dispositif matériel pour chaque morceau de mémoire physique contiguë à l'origine d'un coût d'initialisation important qui limite sa compétitivité pour des messages de taille moindre. Pour de très larges messages, I/OAT réduit la consommation processeur et la pollution de cache, et améliore considérablement les performances globales avec un facteur atteignant 2,5 par rapport au double-buffering.

Nous avons pu observer des résultats similaires sur la machine Idkonn, avec un intérêt du déport de copie au delà de 4 Mo avec le partage du cache L3 (16 Mo) ou de 8 Mo en l'absence de cache partagé. Une étude complémentaire sur des processeurs quadri-cœurs [10] avec des caches partagés de 6 Mo montre les mêmes effets avec des seuils de 50% supérieurs à ceux de la machine Bill (dotée de caches de 4 Mo). Ces résultats révèlent une corrélation entre le seuil à partir duquel déporter les copies sur I/OAT devient intéressant, la taille du cache partagé, et le nombre de processus qui l'utilisent, résumée par la formule :

$$\text{Transfert I/OAT}_{\text{min}} = \frac{\text{taille du cache}}{2 \times \text{nombre de processus utilisant le cache}}$$

10. Harpertown X5460 cadencés à 3,16GHz

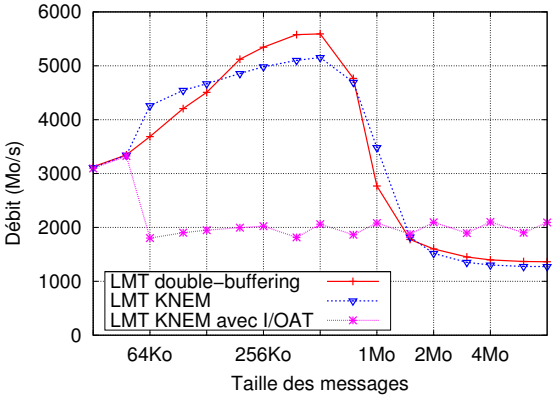


FIGURE 6.16 – *Débit des stratégies double-buffering et* KNEM *(avec et sans* I/OAT*) pour un ping-pong avec un cache partagé L2 de 4 Mo (machine Bill).*

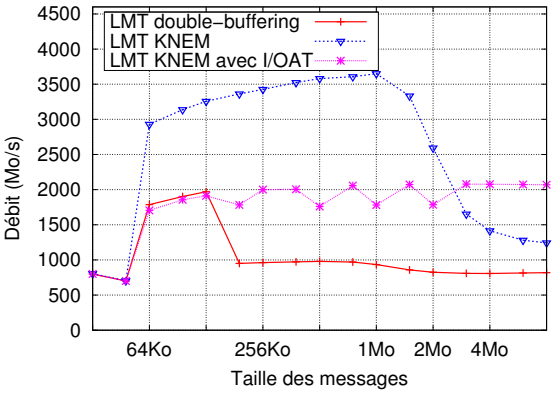


FIGURE 6.17 – *Débit des stratégies double-buffering et* KNEM *(avec et sans* I/OAT*) pour un ping-pong sans cache partagé (machine Bill).*

En effet, une copie directe de KNEM initiée par le processeur impose deux accès au cache par le processus récepteur : les données à émettre doivent être lues en mémoire et placées dans les registres du processeur, puis écrites dans le tampon de réception, chaque opération impliquant un remplissage du cache. Pour éviter que le transfert ne pollue intégralement le cache local, celui-ci doit être au moins deux fois plus grand que la taille du message transféré. Dans le cas contraire, la chute drastique de performances valorise l'utilisation du déport de la copie par I/OAT, qui n'utilise aucune ligne de cache.

Lorsque la stratégie KNEM classique ne bénéficie pas de la réutilisation des données contenues dans le cache (*offcache* activé), ses performances sont largement affectées, au contraire de celles exhibées grâce au déport de copie I/OAT indépendant de l'utilisation des caches. En conséquence, l'interface LMT (normalement utilisée dès 64 ko) bénéficie immédiatement de l'utilisation du déport de copie qui assure une amélioration supérieure à 20% par rapport au LMT double-buffering, comme illustré par la Figure 6.18.

Si l'utilisation du moteur DMA d'I/OAT présente un intérêt certain sur nos machines relativement anciennes Bill et Idkonn, la plateforme Hannibal présente un comportement différent avec des copies directes à l'initiative du processus largement plus performantes que celles associées au déport de copie I/OAT (indépendamment de la taille des messages et des options de transfert [11]). Nous supposons que cela est dû à la stagnation du développement du matériel I/OAT comparé au développement de la technologie d'interconnexion QPI de cette architecture qui assure un tout autre niveau de performance que les bus utilisés dans les machines Bill et Idkonn [12]. De plus, tandis que les performances de la copie I/OAT ne dépendent pas du placement des processus à l'intérieur d'un nœud NUMA, elles sont affectées par les aspects NUIOA impliqués par la distance du

11. Exemples disponibles en Annexe B, Figures B.5 et B.4

12. En effet, tandis que les performances brutes des transferts I/OAT sont passées d'environ 2 à 3 Go/s entre 2007 et 2010, l'introduction de QPI a radicalement multiplié les performances des copies mémoire, poussant par exemple le débit des copies du test Stream [30] autour de 10 Go/s alors qu'elles n'excèdaient guère 3 Go/s sur les architectures dépourvues de cette technologie d'interconnexion.

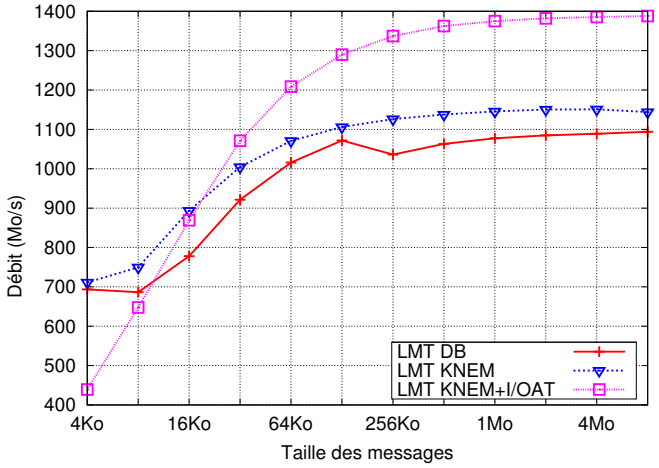


FIGURE 6.18 – *Pingpong IMB entre 2 processus partageant un cache L3 de 16 Mo sur l'architecture Idkonn (offcache activé).*

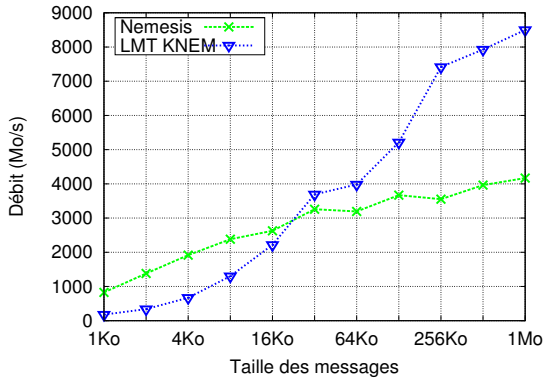
contrôleur d'entrées-sorties (voir Section 4.1.4), et il n'existe actuellement aucun moyen simple de spécifier le moteur DMA utilisé lors du déport de copie.

Seuil d'activation de l'interface LMT

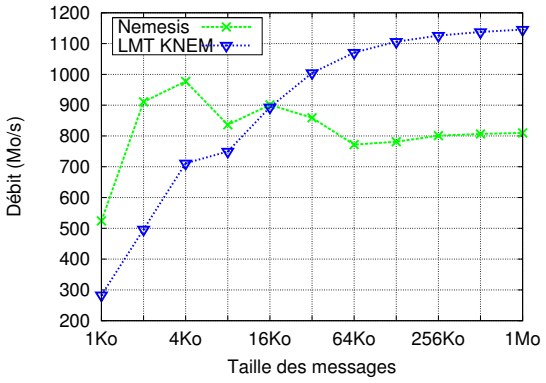
Une question attachée aux performances du sous-système de communication NEMESIS concerne le seuil d'activation du LMT. Le passage de la stratégie employée pour les petits messages [64] à l'utilisation de cette interface est par défaut fixé à 64 ko. Si ce seuil apparaît relativement approprié pour les stratégies vmsplice et double-buffering, nous avons pu observer que l'utilisation du module noyau KNEM peut être profitable pour des tailles de messages inférieures. En effet, le transfert des petits messages s'appuie sur l'utilisation de tampons en mémoire partagée nécessitant deux copies pour chaque échange (Figure 6.2 (a)) et la réduction du nombre de copies offerte par le module spécifique KNEM apparaît généralement avantageuse dès 16 ko [13], comme illustré en Figure 6.19. La modification dynamique de ce seuil dans le module de communication NEMESIS est cependant délicate. Celui-ci a été défini lors de la conception de NEMESIS pour répondre aux particularités d'architectures qui étaient alors d'actualité, et conjointement avec le développement de certaines spécificités d'implémentation [14]. Le changement de ce seuil est ainsi susceptible de compromettre le fonctionnement de la bibliothèque et pourrait nécessiter des changements plus profonds dans l'implémentation.

13. Pour quelques configurations, la transition optimale apparaît avancée à 8 ko ou reculée à 32 ko. Nous pensons ces seuils corrélés aux différents niveaux de caches utilisés (incluant possiblement les caches de niveau L1 non partagés), mais n'avons toutefois pas pu extraire de méthode permettant de les prédire automatiquement.

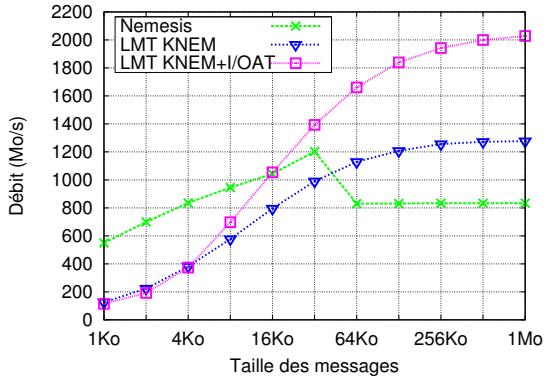
14. L'implémentation des communications intra-nœud et inter-nœuds est étroitement dépendante d'une structure mémoire appelée *cellule* dont la taille est fixée à 64 ko.



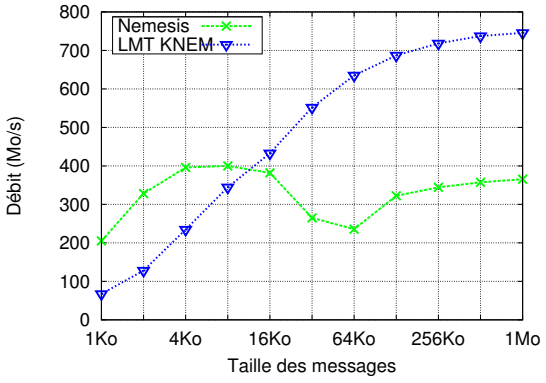
(a) Débits sur Hannibal en l'absence de cache partagé.



(b) Débits sur Idkonn avec un cache L3 partagé (16 Mo).



(c) Débits sur Bill avec un cache L2 partagé (4 Mo), option offcache activée.



(d) Débits sur Bertha pour un placement sur des sockets différents, option offcache activée.

FIGURE 6.19 – *Exemples de transitions entre la stratégie "Nemesis" optimisée pour le tranfert de petits messages et le LMT* KNEM*, pour diverses configurations sur les machines de notre plateforme de test.*

Synthèse des communications point-à-point

Nous avons évalué les différentes stratégies mises à disposition par l'interface LMT de MPICH2-NEMESIS pour le transfert intra-nœud de larges messages. Les performances obtenues pour les communications point-à-point pour différentes architectures et placements de processus montrent des résultats variés. Nous pouvons toutefois en dériver les caractéristiques suivantes :

- En l'absence de cache commun aux cœurs sur lesquels les processus s'exécutent, l'utilisation du module noyau KNEM surpasse largement les autres stratégies et devrait être utilisé [15].
- Dans le cas contraire, les performances de la copie double-buffering et du transfert direct via KNEM exhibent des performances relativement proches, sauf sur la machine 96 cœurs Bertha pour laquelle le double-buffering présente un avantage notable. Nous pensons que ce résultat est corrélé au chipset spécifique IBM et au protocole de cohérence de cache propre à cet hôte.

15. En l'absence de ce module, le LMT vmsplice devrait être privilégié.

– Lorsqu'elle est disponible, la technologie I/OAT offre un déport de copie intéressant sur les architectures ne disposant pas de technologie d'interconnexion récente. Son activation devrait alors être nuancée par la réutilisation des données contenues dans le cache attendue (systématique avec KNEM pour une concurrence forte, à partir d'un seuil proportionnel à la taille des caches dans le cas contraire). Cette fonctionnalité présente également un potentiel intéressant pour le recouvrement des communications puisque la copie est déportée hors du processeur.
– Enfin, l'emploi du LMT KNEM devrait succéder à la copie Nemesis dédiée à l'échange de petits messages à partir de 16 ko.

Ces résultats nous permettent d'implémenter des transitions automatiques entre les stratégies de transfert en fonction du placement des processus et de l'architecture matérielle sous-jacente extraits à l'aide de la bibliothèque `hwloc`. Toutefois, les observations sur ce simple modèle point-à-point ne peuvent être généralisées à des applications réelles qui effectuent des opérations collectives et des transferts point-à-point concurrents.

6.2.3 Opérations collectives

Pour approfondir notre analyse des performances de MPICH2-NEMESIS, cette section présente les résultats obtenus dans le cadre des opérations collectives. Celles-ci sont composées de multiples transferts point-à-point entre paires de processus, chacun appuyé sur le LMTs double-buffering ou KNEM. Comme précisé en introduction, les expériences menées s'appuient sur la suite INTEL MPI Benchmark et invoquent des échanges entre 8, 16 ou 64 processus selon le nombre de cœurs disponibles sur les machines ciblées. Pour chacune, l'option *offcache* des tests IMB [31] est activée.

6.2.3.1 Performance des LMTs

Nous nous attachons dans un premier temps à la comparaison des stratégies de transfert double-buffering et assistées par le noyau grâce au module KNEM. La Figure 6.20 présente les débits agrégés [16] obtenus pour les tests *Alltoall* sur la machine à huit cœurs Bill.

Comme attendu, la réduction de la consommation processeur et de la pollution de cache apportée par KNEM profite largement aux performances. Le gain atteint ici 40% pour la stratégie KNEM classique et plus de 50% avec le déport de copie I/OAT, en comparaison du double-buffering. Les résultats observés sur nos plateformes et pour différents schémas d'opérations collectives [17] confirment bien que KNEM offre une amélioration de performances pour le transfert de larges messages qui est d'autant plus importante que le schéma de communication est intensif. Parmi eux, seule l'opération Reduce dénote d'un bénéfice de la stratégie double-buffering sur l'architecture Bertha (Figure 6.21) qui se démarquait déjà au niveau des communications point-à-point par un cas particulier profitant de ce LMT (voir Table 6.2). Sur les autres architectures, les performances

16. Calculés à partir du total des données transférées pour chaque opération. Le débit présenté pour l'opération Alltoall, qui nécessite l'envoi par les *n* processus intervenant d'un message à chaque autre processus (*n-1*), est ainsi obtenu en divisant le temps d'exécution moyen par $n*(n-1)*taille_message$.
17. Allgather, Allreduce, Reduce, Scatter, Broadcast, Gather.

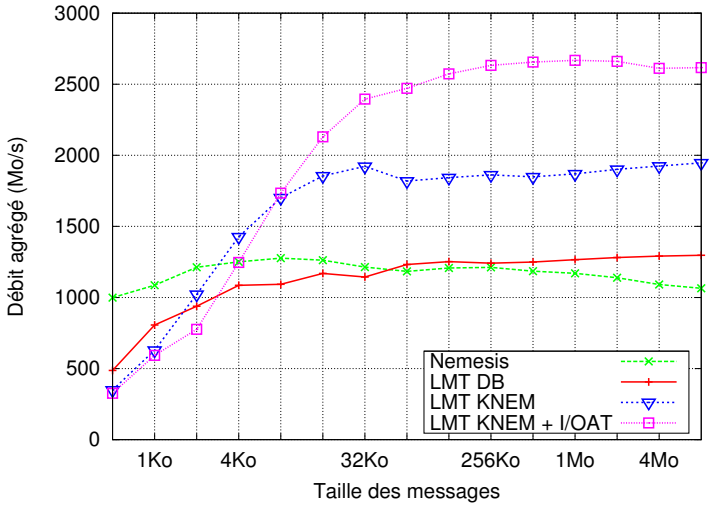


FIGURE 6.20 – *Débits agrégés d'un Alltoall IMB sur la machine Bill pour les stratégies de transfert double-buffering et* KNEM *(8 processus).*

du LMT KNEM pour cette opération sont comparables à celles du double-buffering ou légèrement avantageuses (Figure 6.22).

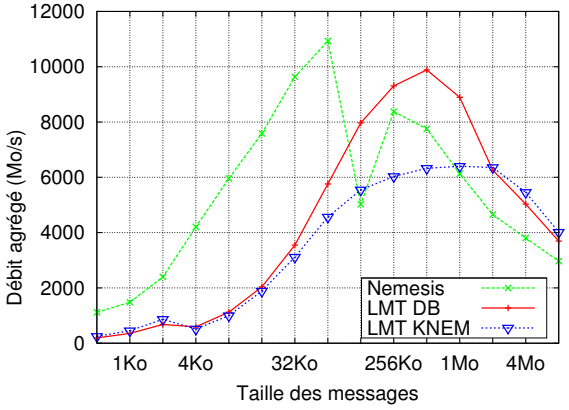


FIGURE 6.21 – *Débits du test Reduce obtenus sur la machine Bertha (64 processus).*

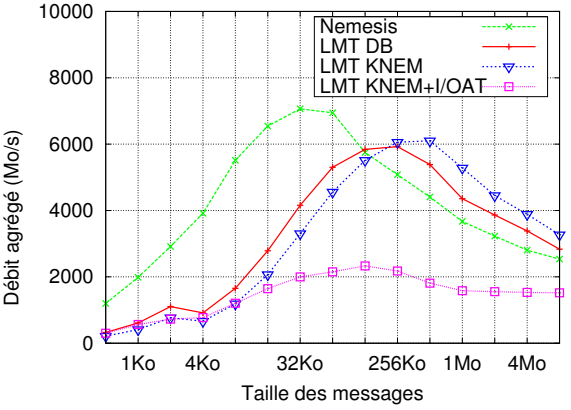


FIGURE 6.22 – *Débits du test Reduce obtenus sur la machine Hannibal (16 processus).*

D'un point de vue général, la stratégie KNEM apparaît bien comme la plus adaptée aux opérations collectives. Un résultat plus intéressant concerne les seuils de transition entre la stratégie dédiée au transfert de petits messages (Nemesis) et KNEM, et les seuils d'activation du déport de copie. En effet, les seuils de transition extraits de l'observation des transferts point-à-point semblent affectés de manière non uniforme par les schémas de communication complexes invoqués dans les opérations collectives. Comparé à la taille de message de 16 ko que nous avions jugée adéquate pour le passage de Nemesis au LMT KNEM, ce seuil apparaît par exemple avancé pour le Alltoall sur Bill (Figure 6.20), et reculé pour les opérations de Reduce (Figure 6.22 et 6.21). Nous proposons ainsi d'observer plus en détails les tailles de messages à partir desquelles les méthodes

devraient idéalement être employées au sein des opérations collectives pour affiner leur utilisation conjointe en fonction des schémas de communication.

6.2.3.2 Étude des seuils de transition

Schéma de communication *all-to-one* : IMB Reduce
En reprenant les courbes proposées pour de débit du test Reduce (Figure 6.22 et 6.21), nous observons que la copie Nemesis est toujours plus rapide que l'utilisation de KNEM pour de petits messages tandis que celui-ci devient plus intéressant pour des messages de taille supérieure à 128 ko. Nous avons pu constater un comportement similaire et un seuil équivalent sur différentes machines et avec un nombre de processus variable. Les opérations de *Allreduce* témoignent de caractéristiques comparables avec un seuil légèrement inférieur [18].

Pour ces opérations, le déport de copie I/OAT n'améliore pas (ou de façon infime) les performances[14], même pour de très larges messages sur les plateformes pour lesquelles il s'avérait largement bénéfique pour les échanges point-à-point. Nous pensons que les copies I/OAT comme les copies directes initiées par le processeur saturent ici le bus mémoire de l'hôte lorsque de multiples processus envoient de larges messages. La bande passante s'en trouve ainsi limitée par la capacité du bus mémoire plus que par l'implémentation de la copie elle-même. Bien que cette fonctionnalité n'améliore pas les performances brutes de ces opérations, il est toutefois important de noter que la portée de ce résultat s'inscrit dans le cadre des opérations collectives bloquantes que nous étudions ici. Elle pourrait profiter dans le cadre d'opérations collectives non bloquantes (ajoutées dans la future révision 3.0 du standard MPI) en améliorant le recouvrement des communications.

Schéma de communication *one-to-all* : IMB Scatter
La Figure 6.23 présente le débit agrégé de 16 processus effectuant un IMB *Scatter* sur la machine Idkonn. Comme observé pour les tests de Reduce, la copie Nemesis apparaît plus performante pour le transfert de messages jusqu'à 128 ko tandis que KNEM est avantageux au-delà de cette taille. Encore une fois, ce comportement et ce seuil varient peu selon les machines de test. Nous avons également observé des comportements semblables sur d'autres opérations one-to-all telles le Broadcast[14], avec un seuil pouvant être avancé à 64 ko.

Le déport de copie I/OAT ne semble pas apporter de bénéfice, sauf sur la machine Bill sur laquelle le débit de très larges messages est augmenté de 10%. Ce phénomène est différent de celui observé précédemment pour le Reduce. Nous pensons que le test Reduce ne peut bénéficier d'I/OAT car toutes les réceptions sont traitées séquentiellement par le même processus récepteur, tandis que Scatter permet à tous les récepteurs de déporter simultanément des copies sur les différents *channels* DMA matériel.

Schéma de communication *all-to-all* : IMB *Alltoall*
Les courbes présentées Figures 6.20, 6.24 et 6.25 présentent les résultats obtenus pour le test Alltoall sur les architectures Bill, Idkonn et Bertha

18. Exemples disponibles en Annexe B.3

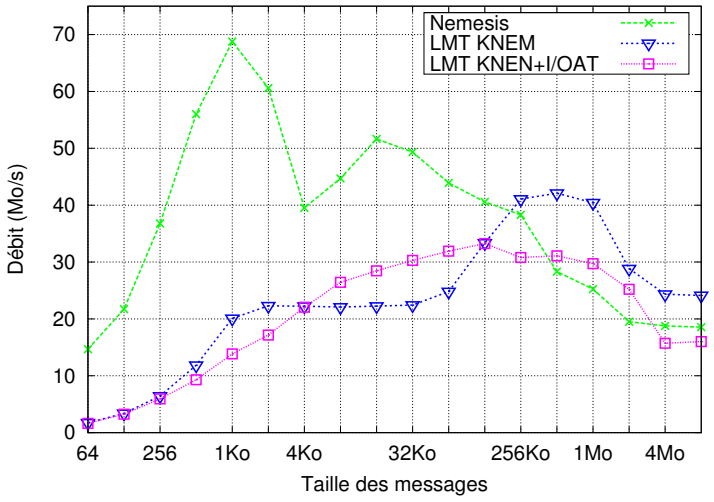


FIGURE 6.23 – *Débits agrégés pour le test IMB Scatter sur la machine Idkonn (16 processus).*

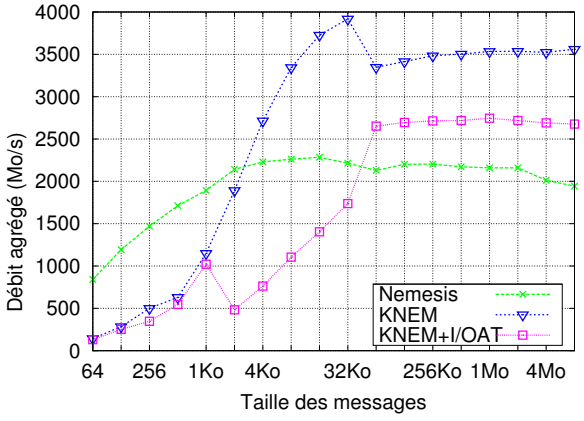


FIGURE 6.24 – *Débits agrégés pour le test IMB Alltoall sur la machine Idkonn (16 processus).*

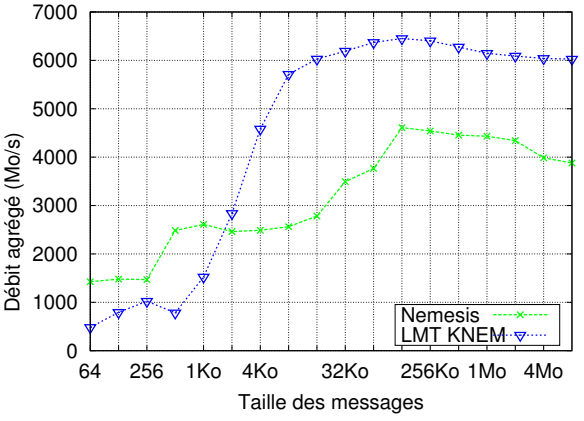


FIGURE 6.25 – *Débits agrégés pour le test IMB Alltoall sur la machine Bertha (64 processus).*

D'un point de vue général, les tests de Alltoall qui présentent le schéma de communication le plus intensif, bénéficient le plus fortement de la réduction du nombre de copies du LMT KNEM [19]. Le seuil d'échange entre les stratégies de communication Nemesis et KNEM apparaît ainsi notablement avancé par rapport au seuil déterminé pour les transferts point-à-point, au contraire des opérations collectives étudiées précédemment. Il apparaît dans la globalité inversement proportionnel au nombre de processus. Cela tend à confirmer le fait que plus le nombre de processus est grand, plus la contention générée est forte et plus la réduction de copie amenée par KNEM est profitable.

Encore une fois, le déport de copie I/OAT est avantageux sur la machine Bill, apportant un gain

19. Avec une amélioration du débit de transfert des messages larges et médium de 50% sur Bertha et d'un facteur 2 sur les autres hôtes.

de débit de 30% (Figure 6.20), tandis que son utilisation diminue les performances obtenues sur les hôtes Hannibal et Idkonn. Il semble donc que l'emploi du déport de copie I/OAT soit indiqué pour des machines anciennes [20] et pour une quantité limitée de transferts simultanés (présentant des contentions réduites sur l'usage du moteur DMA).

Nous avons enfin observé des comportements d'une certaine façon similaires avec d'autres opérations de type all-to-all telles que le Allgather [21], avec une amélioration due à l'utilisation de KNEM moins forte mais aussi moins claire. Ces effets reposent notamment sur le fait que MPICH2 utilise des algorithmes complexes mettant en œuvre des divisions et/ou agrégations de messages qui complexifient l'interprétation des résultats.

Synthèse

Nous avons étudié les performances des stratégies de transfert de larges messages au sein de la bibliothèque MPICH2 pour différents types d'opérations collectives. Notre évaluation témoigne de l'intérêt notable de l'utilisation du module KNEM, capable d'effectuer des transferts directs entre les processus locaux à une machine. Nos expériences montrent qu'il est possible d'affiner l'utilisation conjointe des stratégies de transfert en fonction des schémas de communication invoqués. Si le seuil d'activation du LMT KNEM devrait être associé à une taille de message de 16 ko pour les transferts point-à-point, celui-ci devrait être reculé autour de 128 ko pour des communications de types one-to-all ou all-to-one, ou au contraire avancé pour les communications all-to-all, en fonction du nombre de processus intervenant. De plus, nous avons observé que l'utilisation du déport de copie par la technologie I/OAT n'apparaît intéressante que pour des architectures anciennes et un niveau relativement faible de contention. Son utilisation pourrait toutefois être judicieuse dans le cadre d'opérations collectives non bloquantes en favorisant le recouvrement des copies par le calcul.

L'observation du comportement des opérations collectives nous mène à la conclusion que les transferts de données assistés par le noyau permettent un réel gain de performances dans le cadre des communications MPI intra-nœud, grâce à la réduction du nombre de copies des données conjointe à un modèle spécifiquement adapté au schéma de communication MPI. De plus, les expériences menées utilisent des opérations collectives implémentées au travers d'une composition de transferts point-à-point. Celle-ci induit une séquentialisation du traitement des transferts qui peut ralentir leur progression. Des travaux menés récemment en collaboration avec le laboratoire ICL de l'Université du Tennessee à Knoxville sont venus compléter les nôtres pour adresser ce problème [117]. Ils proposent une implémentation native des opérations collectives sur le module KNEM qui tient compte du schéma collectif lors de la soumission des requêtes KNEM pour en optimiser l'utilisation.

La contribution du module noyau KNEM au sein de MPICH2-NEMESIS est un support indéniable à l'obtention de communications intra-nœud performantes, indispensables à l'exploitation efficace des clusters de calcul contemporains. Pour en attester, nous présentons maintenant ses bénéfices dans un contexte applicatif.

20. Dû au retard de développement de la technologie I/OAT qui limite ses performances en comparaison de celui des technologies d'interconnexion récentes.

21. Exemples disponibles en Annexes B.3.3.

6.2.4 NAS Parallel Benchmarks

Nous avons évalué les performances applicatives des stratégies de transfert double-buffering et KNEM au travers de tests de la suite NAS [29]. La Table 6.3 résume les temps d'exécution des tests parallèles IS et FT [22] sur l'ensemble de nos plateformes expérimentales.

Machine	Benchmark	double-buffering	KNEM	Accélération
Hannibal	ft.B.8	14.60 s	13.90 s	**+5%**
	ft.C.8	63.77 s	60.31 s	**+5.7%**
	is.B.8	0.70 s	0.60 s	**+16.6%**
	is.C.8	2.81 s	2.41 s	**+16.6%**
Bill	ft.B.8	40.43 s	36.02 s	**+12.2%**
	ft.C.8	175.46 s	158.36 s	**+10.8%**
	is.B.8	2.42 s	1.89 s	**+12.2%**
	is.C.8	10.04 s	8.02 s	**+25.2%**
Idkonn	ft.B.16	24.14 s	21.70 s	**+11.2%**
	ft.C.16	97.65 s	85.73 s	**+13.9%**
	is.B.16	1.29 s	0.99 s	**+30.3%**
	is.C.16	5.88 s	4.43 s	**+32.7%**
Bertha	ft.C.64	31.91 s	28.82 s	**+10.7 %**
	ft.D.64	727.37 s	645.36 s	**+12.7 %**
	is.C.64	3.17 s	2.29 s	**+38.4 %**
	is.D.64	65.00 s	52.72 s	**+23.3 %**

TABLE 6.3 – *Temps d'exécution des tests IS et FT de la suite NAS Parallel Benchmarks sur nos différentes plateformes de test.*

Comme attendu, l'utilisation de KNEM améliore largement les performances sur l'ensemble des machines, jusqu'à 38% pour le test IS et 12% pour FT avec 64 processus exécutés sur la machine Bertha. Ces gains apparaissent entre autres corrélés à une réduction significative du nombre de défauts de cache (observable à l'aide de l'outil PAPI [17]). Pour exemple, la Table 6.4 présente le nombre de défauts de cache relevés durant l'exécution du test IS sur la machine Bill en fonction de la stratégie de transfert utilisée [23]. Comparées à la copie double-buffering, la copie noyau KNEM réduit ici le nombre de défauts de cache de plus de 15% et l'utilisation du déport de copie I/OAT de 20%.

	LMT double-buffering	LMT KNEM	LMT KNEM avec I/OAT
is.B.8	11.25 M	9.50 M	8.92 M

TABLE 6.4 – *Défauts de cache observés lors de l'exécution du test IS en fonction des LMTs (machine Bill, 8 processus).*

De plus, le bénéfice observé varie peu selon la classe de test (la taille du problème), indiquant que

22. Les autres tests de cette suite n'utilisent que très peu de gros messages et ne montrent ainsi que de légères variations de performance. Quelques résultats sont proposés pour exemple en Annexe B.4.

23. Des résultats illustrant la réduction du nombre de défauts de cache dans le cadre des opérations point-à-point et collectives sont également disponibles en Annexe B.4.

l'utilisation de transferts intra-nœud directs assistés par le noyau KNEM devrait profiter non seulement aux microbenchmarks, mais également aux applications réelles en réduisant la contention et la pollution de cache.

Nous avons également observé que l'utilisation du déport de copie I/OAT n'améliore pas les performances sur Idkonn et Hannibal, tandis qu'il offre une légère amélioration sur l'architecture Bill (+6,1% pour ft.C.8, +0% pour is.C.8). Ce résultat confirme les observations faites dans les sections précédentes : I/OAT est principalement intéressant pour les opérations point-à-point ou impliquant de faibles contentions sur des architectures relativement anciennes (pour lesquelles les copies initiées par le processeur sont particulièrement lentes, comme introduit en Section 6.2.2).

6.3 Bilan

La généralisation des processeurs multicœurs a conduit à une sévère hiérarchisation des nœuds de calcul employés au sein des grappes avec de multiples cœurs, caches partagés et nœuds NUMA. Les communications intra-nœud sont devenues critiques sur les performances, et l'assistance du noyau est une solution intéressante pour améliorer leur efficacité. En collaboration avec l'équipe Radix du laboratoire d'Argonne, nous avons enrichi l'implémentation MPICH2 du standard MPI du module noyau dédié KNEM. Celui-ci permet d'effectuer des transferts de données directs entre les processus en mode synchrone ou asynchrone, et dispose d'un support permettant de profiter des fonctionnalités de déport de copie mémoire intégrées au matériel I/OAT d'INTEL.

Nous avons mené une étude approfondie des performances de transfert intra-nœud soutenues par le sous-système de communication NEMESIS de MPICH2 au niveau des opérations point-à-point et collectives, et de tests applicatifs sur différentes machines multicœurs comprenant trois générations de plateformes INTEL, et avec un maximum de 96 cœurs sur un unique nœud à mémoire partagée. Nos résultats montrent que la réduction du nombre de copies mémoire avec l'assistance du noyau est effectivement une solution intéressante pour le transfert de larges messages, notamment en comparaison de la stratégie double-buffering effectuée en espace utilisateur et pour laquelle l'emploi de tampons intermédiaires en mémoire partagée nécessite de multiples recopies. Grâce à la diminution de l'utilisation du processeur et d'une meilleure utilisation du cache, le LMT KNEM apporte ainsi une large augmentation de débit pouvant atteindre un facteur 2 et jusqu'à 30% d'accélération sur certains tests de la suite NAS, même sur de larges machines NUMA.
Ces travaux sont apparentés aux propositions faites dans la bibliothèque MVAPICH qui disposent d'approches spécifiques pour les transferts intra-nœud (notamment de copie directe à l'aide du module noyau LIMIC2 [82]). Leur nombre est cependant limités en comparaison des solutions implémentées au sein de MPICH2-NEMESIS (2 stratégies dans MVAPICH contre 4 dans MPICH2) et les expériences menées sont restreintes à des architectures relativement anciennes, notamment dépourvues des nouvelles technologies d'interconnexion et de structure NUMA. Le bénéfice indiscutable et les fonctionnalités du module noyau KNEM ont également séduit les développeurs d'autres implémentations MPI. KNEM a ainsi été intégré au sein de la bibliothèque OPEN MPI 1.5 .

Nous avons observé que les performances du module noyau KNEM souffrent moins du place-

ment que le traditionnel modèle double-buffering, et souligné l'importance de calculer les seuils d'échange entre les stratégies pour affiner leur utilisation conjointe. Dans l'idéal, l'activation des LMTs de MPICH2-NEMESIS devrait être déterminée dynamiquement en fonction de la tailles des données échangées, du partage de ressource entre les tâches communicantes et du schéma de communication mis en œuvre. Pour ce faire, il est nécessaire d'une part de sélectionner la stratégie la plus adaptée (double-buffering, KNEM avec ou sans I/OAT, ou à défaut vmsplice) aux transferts de larges messages et d'autre part de déterminer le seuil à partir duquel le LMT devrait effectivement succéder aux méthodes de transfert des petits messages, selon le contexte de communication (transferts point-à-point ou schéma d'opération collective).
L'impact de la topologie sur les performances des stratégies de transfert et le problème de leur sélection font également l'objet de travaux dans d'autres bibliothèques MPI. Ils ont par exemple été étudiés dans le cadre des transferts soutenus par l'implémentation MVAPICH [83], et les contributeurs du projet OPEN MPI offrent par exemple de paramétrer l'emploi de diverses stratégies en fonction de la topologie et du partage de caches dans les architectures [72].

Nous avons mis en évidence certains comportements complexes dans le cadre des opérations collectives qui complexifient l'ajustement du choix de la meilleure stratégie de communication. Les seuils de transition apparaissent relativement approximatifs et l'idée de faire descendre des informations sur les opérations collectives dans le LMT (voire dans le module KNEM) nécessite d'être étudiée mais elle pourrait entraîner une manipulation des données coûteuse. Nous avons également montré que le déport de copie I/OAT n'apporte actuellement un gain que pour des machines anciennes et souffrant de performances mémoire pauvres, bien qu'il puisse toutefois profiter au recouvrement des communications MPI non bloquantes.

L'optimisation des opérations collectives grâce au module noyau KNEM a depuis été approfondie. KNEM propose désormais d'une interface native adaptée aux schémas de ces opérations qui accroît encore l'amélioration des communications MPI en mémoire partagée et contribue à l'élaboration d'opérations adaptées aux structures hiérarchiques des nœuds de calcul [117].

Conclusion

La généralisation des processeurs multicœurs a conduit à la multiplication des ressources partagées entre les unités de calcul, qui profitent de structures de cache multiniveaux, partagent de la mémoire ou encore l'accès à divers périphériques tels que les cartes réseau ou les accélérateurs GPU. Avec le nombre croissant de cœurs, les constructeurs ont dû renoncer aux bus mémoire centralisés au profit de nouvelles technologies d'interconnexion. Celles-ci assurent un meilleur passage à l'échelle mais ont réintroduit les architectures à accès mémoire non uniforme (NUMA). Les machines de calcul modernes sont ainsi caractérisées par une forte hiérarchie interne constituée de plusieurs nœuds NUMA, comprenant eux-mêmes des processeurs dotés de multiples cœurs et d'une structure arborescente de caches. Cette organisation est à l'origine d'affinités matérielles entre les composants et dont une mauvaise gestion impacte grandement les performances. Dans le domaine du calcul haute performance qui requiert des besoins toujours croissants de puissance de calcul, ces différents niveaux de topologie matérielle sont autant de niveaux de complexité qui doivent être pris en compte pour exploiter les architectures parallèles efficacement.

Les différents modèles de programmation mis au point au cours des précédentes décennies ont été conçus pour des architectures plates sur lesquelles les accès uniformes aux ressources permettaient leur exploitation de façon relativement harmonieuse. En l'absence de consensus sur un modèle réellement adapté aux architectures multicœurs, ces modèles de programmation sont utilisés pour programmer les architectures parallèles contemporaines. Ils se heurtent au problème complexe des affinités matérielles qui varient selon le matériel et les constructeurs et exhibent des effets parfois difficiles à prédire.

Exploiter la quintessence des architectures modernes repose ainsi sur une prise en compte fine de ces structures hiérarchiques. Les difficultés engendrées et le niveau d'expertise requis pour cette entreprise ne permettent pas de confier cette tâche aux programmeurs d'applications. De plus la portabilité des applications sur les différentes topologies disponibles sur le marché qui ne cessent de se renouveler, doit être assurée. C'est ainsi au niveau des supports exécutifs que devrait se faire la prise en charge de ces structures, de sorte que ceux-ci puissent assurer une bonne gestion de celles-ci ainsi que la portabilité des performances.

Contribution

Les travaux de cette thèse proposent tout d'abord une évaluation des aspects de localité des périphériques sur les performances des transferts sur réseau rapide dans les grappes de calcul. En

effet, les architectures NUMA présentent non seulement des temps d'accès variables aux différents bancs mémoire, mais sont également caractérisées par des accès non uniformes aux bus d'entrées-sorties attachés sur certains nœuds NUMA. Il en résulte des variations de débit potentiellement importantes susceptibles de diminuer les performances de transfert lorsque l'accès aux interfaces réseau est effectué depuis un nœud NUMA distant. Pour contrebalancer ces effets NUIOA (*Non Uniform Input/Output Access*), nous avons proposé d'adapter la stratégie de placement des tâches au sein de la bibliothèque NEWMADELEINE, destinée à l'exploitation des réseaux haute performance. Notre méthode assure le placement des processus communicants à proximité des interfaces réseau. Elle permet ainsi d'obtenir des performances maximales et reproductibles sans intervention de la part de l'utilisateur. Cette implémentation repose sur une détection automatique des périphériques, qui a été extraite et intégrée dans la bibliothèque `hwloc` qui fournit un ensemble d'outils indispensables à l'exploitation des architectures contemporaines.

Une façon complémentaire de prendre en compte la localité des interfaces réseau consiste à optimiser non pas le placement des processus mais les stratégies de communication en fonction de la position physique des cartes et des cœurs sur lesquels sont exécutées les tâches communicantes. Nous nous sommes ainsi intéressés à l'intégration des contraintes NUIOA au niveau des stratégies de transfert mises en œuvre au sein des bibliothèques MPI dans le cadre d'un placement des processus prédéfini, par exemple pour assurer les performances et leur reproductibilité. Dans ce contexte, les améliorations possibles consistent à adapter la façon dont un processus exploite les cartes réseau, ou au contraire à définir à quels processus devrait revenir la charge d'effectuer des échanges avec une interface donnée.

Nous avons tout d'abord proposé de corriger l'utilisation de multiples cartes réseau en modifiant la répartition des données au travers de chacune en fonction de la localité du processus communicant relative aux interfaces employées [1]. Nous avons mis en évidence que les effets NUIOA intervenant au cours de transferts multirails peuvent être compensés en privilégiant le transfert des données sur l'interface locale en fonction des bandes passantes soutenues et du schéma de communication. Nous avons ainsi intégré au sein de la bibliothèque OPEN MPI une stratégie de répartition NUIOA des données sur les cartes qui permet d'améliorer les performances des communications multirails pour les transferts point-à-point et les opérations collectives.

Nous avons également proposé de modifier la sélection des processus employant une carte donnée dans le cadre de communications collectives hiérarchiques. Celles-ci sont appuyées sur l'utilisation de processus *leader*, en charge d'effectuer les transferts réseau pour l'ensemble des tâches d'un nœud de calcul. Les processus leaders sont ainsi fortement soumis aux aspects NUIOA et devraient donc disposer d'un accès privilégié aux cartes réseau. Nous avons modifié le choix des processus leader sur les nœuds distants de la racine de l'opération pour désigner en tant que tels les processus placés à proximité des cartes, et démontré le potentiel de notre stratégie sur l'implémentation hiérarchique de l'opération de broadcast de la bibliothèque OPEN MPI.

Aujourd'hui, le défi des communications s'est déplacé à l'intérieur des nœuds de calcul, tant le nombre d'échanges locaux augmente avec le nombre de cœurs. Nous nous sommes ainsi attachés à l'étude des transferts intra-nœud, soumis à la topologie hiérarchique des machines de calcul. Leur optimisation a conduit à l'élaboration de multiples stratégies de transferts en mémoire partagée. La collaboration de notre équipe de recherche avec l'équipe Radix du laboratoire d'Argonne a

1. Stratégie à nouveau basée sur la détection de la localité NUIOA extraite de la bibliothèque `hwloc`.

contribué à cette tâche par la mise au point du module noyau KNEM, qui permet d'effectuer des copies directes avec l'assistance du noyau et bénéficie de l'utilisation de la technologie I/OAT des processeurs INTEL. L'ajout de cette stratégie de communication au sein de l'implémentation MPICH2 nous a permis de revisiter l'emploi conjoint des différentes méthodes de communication disponibles. Nous avons ainsi étudié les performances de chacune sur différentes plateformes et analysé l'impact de la topologie sur leurs performances respectives. Nous avons extrait de cette étude des propriétés liées aux caractéristiques de topologie des architectures, et la nécessité d'automatiser finement le passage d'une stratégie à l'autre en fonction de ces particularités matérielles et des spécificités du transfert.

L'ensemble de ces travaux contribue à l'ajustement dynamique des communications en fonction des structures topologiques caractéristiques des machines de calcul modernes, et s'articulent autour de :
- l'adaptation du placement des tâches relatif à la position des cartes pour les transferts réseau,
- l'optimisation des transferts réseau en accord avec la distribution des tâches (en ajustant l'utilisation des cartes ou le choix des processus communicants),
- et l'optimisation des transferts en mémoire partagée en accord avec la distribution des tâches.

Ces travaux concourent ainsi à l'exploitation efficace des architectures contemporaines par les supports exécutifs dont le rôle est de fournir de bonnes performances et leur portabilité tout en masquant la complexité des architectures aux programmeurs d'applications.

Perspectives

Les pistes pouvant être explorées à la suite de ces travaux concernent une meilleure intégration des données de topologie statiques avec les schémas applicatifs ou des modèles de performances permettant d'affiner les stratégies de placement et de transfert. Elles comprennent également l'observation de l'évolution des architectures attendues pour les années à venir, qui devraient restructurer les aspects de localité connus dans les machines contemporaines.

Un meilleur dialogue entre les couches logicielles

Une méthode pour favoriser l'adéquation entre les structures applicatives et matérielles consiste à améliorer les échanges entre les applications et les supports exécutifs. Pour ce faire, il est nécessaire de faire remonter des informations sur la topologie, exploitables au niveau de l'application. En contrepartie, celle-ci doit pouvoir transmettre des indices sur son schéma algorithmique, en indiquant notamment les affinités et leurs éventuelles modifications au cours de l'exécution si plusieurs schémas algorithmiques différents s'enchaînent. Ceci permettrait aux bibliothèques de prendre des décisions adaptées quant au choix du placement des tâches et des données, ou de l'optimisation des stratégies de transfert au fil de l'eau.

Cette piste de recherche n'est pas nouvelle et a été exprimée par de nombreux contributeurs à l'élaboration de supports exécutifs. Elle repose cependant sur l'extension (ou le renouveau) des standards de programmation existants pour y intégrer les supports nécessaires à la transmission des informations de l'application vers les couches logicielles inférieures ou inversement. Cette idée fait l'objet de discussions au sein des groupes de travail attachés à l'évolution des standards tels que

le *MPI-3 Hybrid Working Group* qui travaille à l'élaboration de la prochaine version du standard MPI, ou le OPENMP *Architecture Review Board* attaché à l'évolution d'OPENMP. Une difficulté soulevée par l'extension des standards eux-mêmes dans ce sens concerne la portabilité. En effet, il est difficile de prévoir l'évolution des architectures à venir, et donc les niveaux de topologie qui seront concrètement utiles d'ici quelques années [2]. Les débats menés ne font ainsi pas encore l'unanimité et les informations de topologie envisagées apparaissent encore limitées [3]. En attendant un réel consensus sur l'évolution des standards de programmation, une idée de développement repose sur l'association entre des modèles de performances et des informations de topologies statiques permettant de guider un meilleur ordonnancement des tâches exprimées par ces modèles et l'optimisation des stratégies de communication.

Une approche globale pour l'adaptation du placement et des stratégies de transfert

Si nos travaux se sont axés autour du placement des tâches et des données adapté à la topologie des calculateurs, d'une part, et d'autre part autour de l'adaptation des stratégies de communication en fonction de ces caractéristiques, une solution idéale consisterait a traiter ces propositions conjointement, voire même en fonction du temps lorsque les schémas applicatifs évoluent avec celui-ci. Un tel problème, catégorié comme NP-difficile, ne peut bien sûr pas être abordé légèrement. Néanmoins, si une gestion parfaite est difficilement concevable, la mise en œuvre de stratagèmes permettant de faire collaborer astucieusement la gestion du placement et des stratégies de transfert peut être envisagée. Une proposition pourrait être de considérer tout d'abord un placement des tâches jugé adapté au début de l'application, puis d'en extraire un ajustement des stratégies de communication. L'aspect délicat de cette entreprise consisterait alors à pouvoir quantifier au cours de l'exécution à quel point celles-ci sont appropriées et juger de la pertinence du placement pour éventuellement effectuer des réajustements, soit en modifiant les stratégies de communication, soit en migrant des tâches. Plutôt que de mettre en œuvre une migration réelle des processus, coûteuse et difficile dans un environnement distribué, une alternative consiste à réordonner (*reordering*) les processus pour échanger leurs rôles respectifs sans avoir à les migrer [4]. Si l'application décrit les contraintes de son schéma algorithmique voire indique quand celles-ci changent au cours de l'exécution, les bibliothèques pourront adapter dynamiquement les stratégies de communication lors du réordonnancement des processus. Le potentiel de cette adaptation dynamique doit cependant faire l'objet d'études de coût pour ne pas pénaliser les performances globales par le calcul des ajustements nécessaires. Il devrait également reposer sur une analyse qualitative des transferts générés (poids de l'empreinte mémoire entre les tâches) pour répondre au discernement nécessaire quant à la justesse du placement et des stratégies employées.

Une prise en compte quantitative et qualitative de la topologie

La topologie des architectures a un impact crucial sur les performances des applications. Outre

2. Les niveaux nœuds NUMA, socket, cache partagé et cœurs pourraient ne pas être suffisants voir obsolètes pour caractériser les machines *many-core* attendues pour succéder aux architectures contemporaines.

3. Si le standard MPI dispose par exemple de données relatives à la topologie réseau, dans l'état actuel de son développement MPI-3 ne devrait pas intégrer de notion liée à la localité des interfaces réseau.

4. Comme proposé dans l'équipe Runtime par les travaux d'Emmanuel Jeannot et de Guillaume Mercier [61]. Cette méthode requiert une contribution de l'application. Par exemple, elle peut initier un reordering chaque changement notable attendu sur son schéma de communication grâce à l'appel de fonctions dédiées.

l'organisation structurelle des niveaux de topologie, la conception matérielle même des composants employés dans les architectures influence de façon notable les effets obtenus. Les liens d'interconnexion mémoire telles que AMD HYPERTRANSPORT et INTEL QUICKPATH INTERCONNECT, par exemple, ne proposent pas une gestion des contentions tout à fait équivalente. Nous avons pu observer que l'emploi de chipsets de constructeurs différents pour assembler des processeurs pourtant identiques altèrent sensiblement le comportement des transferts de données [5], ou encore que la façon dont les niveaux de caches sont intégrés au sein des puces peut modifier les caractéristiques d'accès mémoire [6]. La proportion des effets NUIOA et leurs aspects asymétriques semblent également varier pour des technologies réseau et d'interconnexion pourtant relativement proches.

De tels détails ne sont pas pris en compte dans les niveaux de topologie structurels sur lesquels nous avons appuyés nos travaux, bien que déterminants sur les performances attendues, et devraient pouvoir être détectés dynamiquement. Une idée intéressante serait ainsi de pouvoir fournir une *abstraction* de la topologie dotée d'informations *quantitatives* et *qualitatives* sur le partage de ressources afin de mieux appréhender le comportement du matériel lors des transferts de données. Cette abstraction pourrait être développée à partir de modèles de micro-architectures décrivant le comportement de ces partages à partir de mesures de performances bas-niveau (transfert de ligne de cache, transition dans l'automate de cohérence mémoire, ...) comme entrepris dans l'équipe Runtime dans le cadre des travaux de thèse de Bertrand Putigny. Une telle abstraction permettrait d'affiner la prise en compte statique de la topologie des architectures en ajoutant des *annotations* supplémentaires au modèle, voire en effectuant des corrections dynamiques.

De manière orthogonale, les modèles de performances peuvent rester plus haut-niveau et venir en complément des informations de topologie statiques que nous utilisons déjà. Le support exécutif StarPU[28] développé au sein de l'équipe Runtime dans le cadre de la thèse de Cédric Augonnet propose par exemple un ordonnancement performant des tâches sur architectures hétérogènes (constituées de processeurs et d'accélérateurs) guidé par des modèles de coût. Ceux-ci reposent sur un enregistrement de l'historique des exécutions et sur des mesures de performances des transferts des données. Ce modèle se révèle très efficace en pratique sans nécessiter des modèles de coût très précis. Cette idée pourrait être réutilisée pour tenter de prédire les performances des communications et ainsi aider la sélection dynamique des stratégies, même quand l'analyse des performances de transfert ne peut pas être effectuée dans de bonnes conditions [7].

Généraliser pour mieux anticiper

La continuité de nos travaux encadre naturellement l'étude des architectures nouvelles ou à venir. Le maintien de supports exécutifs adaptés à l'exploitation des machines de calcul nécessite en effet de prendre en compte les caractéristiques matérielles apportées par les différentes générations de technologie, et repose sur une bonne compréhension de leurs effets. De plus, la tendance à la miniaturisation et à l'intégration de composants au sein des puces annoncent de sérieux bouleversements dans l'organisation des futures générations de machines.

5. Les chipsets INTEL et IBM des machines Idkonn et Bertha (processeurs identiques Dunnington Xeon) semblent ainsi modifier les performances obtenues pour les stratégies de transfert intra-nœud de MPICH2 (voir Section 6.2.2).

6. Justifiant l'avantage du double-buffering dans certains cas sur les processeurs Dunnington (Section 6.2.2).

7. Par exemple dans le cas d'un partage des ressources avec des applications concurrentes susceptible de fausser les mesures de performances.

L'intégration de composants tels que les contrôleurs mémoire ou d'entrées-sorties au sein des puces réordonnent dès à présent les structures classiques des topologies. Par exemple, les processeurs *Magny-Cours* développés par AMD juxtaposent des puces hexa-cœurs embarquant chacune un contrôleur mémoire au sein d'un même socket. Chaque demi-socket peut ainsi disposer de son propre banc mémoire abaissant la structure de nœud NUMA à l'intérieur du socket, ce qui inverse ici ces niveaux hiérarchiques ! Le marché s'oriente également vers la distribution de puces intégrant des unités de calcul de natures différentes. Le projet AMD Fusion propose par exemple des *Accelerated Processing Unit* (APU) associant CPU et GPU sur une unique puce et une proposition voisine a été développé par INTEL sur les puces Sandy-Bridge. De telles structures peuvent ainsi créer des affinités entre ces composants logiques radicalement différents et susceptibles d'affecter le partage des ressources. La présence de cache partagé entre ces composants [8] nécessiterait par exemple d'analyser l'impact de la concurrence des GPUs et CPUs qui disposent de modèles mémoire très différents.

De plus, les spéculations sur les architectures du futur mentionnent des puces comprenant plusieurs dizaines de cœurs organisés sous la forme de grilles [9] (voire de tores) et la perte de la cohérence de cache dans les systèmes [14]. De telles spécificités devraient chambouler la notion de localité en ajoutant par exemple un niveau intermédiaire entre mémoire partagée et mémoire distribuée et pourraient mener à l'ébranlement des modèles de programmation actuels déjà limités. Un tel niveau intermédiaire pourrait bénéficier au modèle de programmation MPI alors que son utilisation était dans une certaine mesure remise en cause au sein des nœuds de calcul, mais nécessiterait de revisiter les stratégies de transfert proposées. Il constituerait au contraire une remise en cause critique des modèles de programmation appuyés sur la mémoire partagée tels que le multithreading ou l'emploi d'OPENMP.
C'est ainsi à une nouvelle révolution que les constructeurs vont confronter les programmeurs. Il conviendra alors de proposer des solutions adaptées à de telles caractéristiques, voir de repenser les supports exécutifs pour qu'ils puissent administrer convenablement de telles structures et éviter de creuser le ravin qui sépare déjà les performances soutenues des performances théoriques.

8. Notamment sur les processeurs Sandy-Bridge.
9. Notamment entrepris dans le cadre du programme de recherche Tera-scale d'INTEL [15].

Annexes

Annexe A

Plateforme de tests

A.1 Dalton/Dalton-Infini

Les machines *Dalton* (Figure A.1) sont des architectures bi-processeurs bi-cœurs AMD OPTERON 265 [1] cadencés à 1.8 GHz, dotées de deux bancs de 512 Mo de RAM NUMA. Elles possèdent deux interfaces réseau, MYRI-10G et QSNET II (ELAN4) connectées sur le nœud NUMA 0 respectivement au travers de bus PCIE et PCI-X.

Les machines *Dalton-Infini* (Figure A.2) ne diffèrent des Dalton que par leur interface réseau. Chacune possède uniquement une carte INFINIBAND DDR (MELLANOX INFINIHOST III MT25208) attachée au nœud NUMA 1.

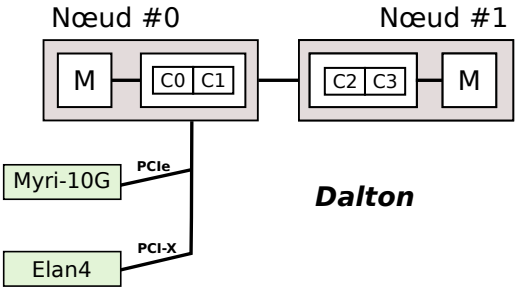


FIGURE A.1 – *Bi-processeur bi-cœurs* OPTERON *Dalton.*

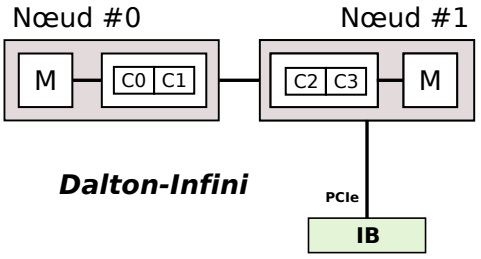


FIGURE A.2 – *Bi-processeur bi-cœurs* OPTERON *Dalton-Infini.*

1. Interconnectés par liens HYPERTRANSPORT à 1 GHz (génération 1.0)

A.2 Hagrid

Hagrid (Figure A.3) est une plateforme NUMA octo bi-cœur AMD OPTERON 865[1] cadencés à 1 GHz. Elle dispose d'une carte INFINIBAND DDR (MELLANOX INFINIHOST III MT25208) attachée au nœud NUMA 0.

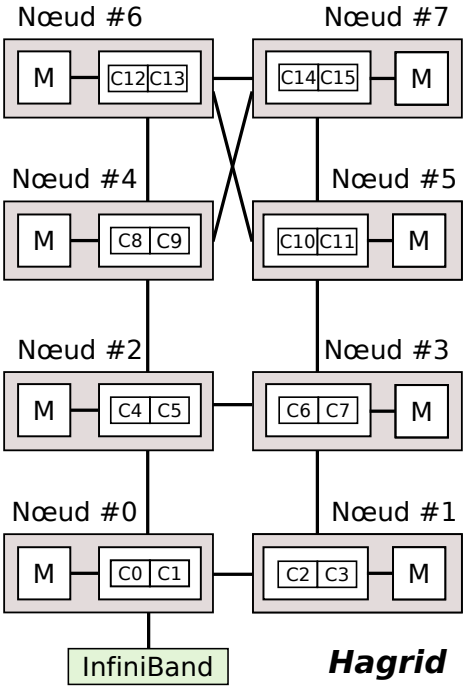


FIGURE A.3 – *Octo bi-cœur* OPTERON *Hagrid.*

A.3 Dancharia

La machine *Dancharia* est une architecture à 8 nœuds NUMA composés d'hexa-cœurs AMD OPTERON 8439-SE[2] (Figure A.5) cadencés à 2,8 GHz. Elle est dotée de 128 Go de RAM et dispose d'une carte INFINIBAND DDR (MELLANOX ConnectX MT26448) reliée au nœud NUMA 0 et d'une carte Gigabit Ethernet (NetXtreme II BCM5706) (Figure A.4).

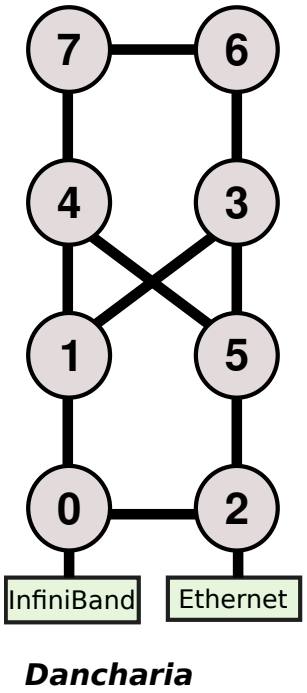


FIGURE A.4 – *Topologie* NUMA *de la machine Dancharia.*

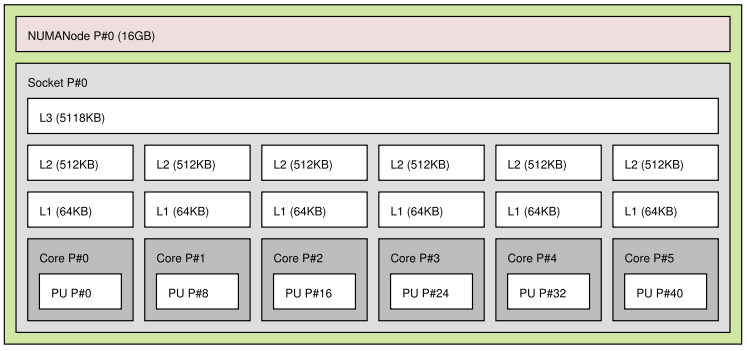


FIGURE A.5 – *Organisation topologique d'un nœud* NUMA *de la machine Dancharia exposée par* `hwloc`.

2. Interconnectés par liens HYPERTRANSPORT version 3.0 (2.4 GHz)

A.4 Borderline

La plateforme *Borderline* est composée de huit machines quadri-puces bi-cœurs AMD OPTERON 8218[1] cadencés à 2.6 GHz. Chaque hôte possède quatre nœuds NUMA dont les nœuds 0 et 1 sont connectés à leur propre bus d'entrées-sorties. Chacun d'entre eux dispose d'un connecteur PCIE 8x relié à une carte MYRICOM MYRI-10G ou INFINIBAND DDR (MELLANOX ConnectX MT25418). Ces hôtes possèdent ainsi deux interfaces MYRI-10G, deux interfaces INFINIBAND, ou une interface de chaque technologie.

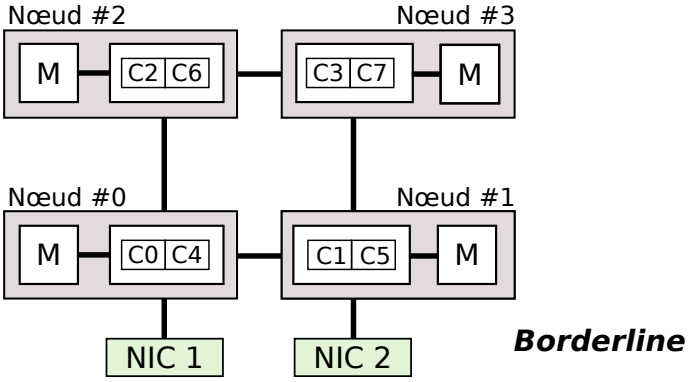


FIGURE A.6 – *Quadri bi-cœur* OPTERON *Borderline.*
Deux interfaces réseau sont connectées aux nœuds NUMA *0 et 1.*

A.5 Mirage

Les machines du cluster *Mirage* sont composées de bi hexa-cœurs INTEL Westmere Xeon X5650 cadencés à 2,67 GHz interconnectés au travers de la technologie QPI et dotées de 36 Go de RAM. Chacune dipose de trois GPU NVIDIA Tesla M2070 connectés sur les deux nœuds NUMA, et d'une carte INFINIBAND QDR (MELLANOX ConnectX MT26438) attachée au nœud 0 (Figure A.7).

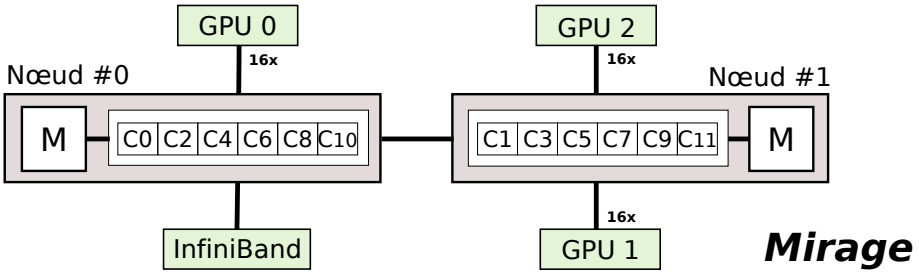


FIGURE A.7 – *Bi hexa-cœur* INTEL *Westmere Mirage.*

A.6 Hannibal

La machine *Hannibal* est une architecture bi-processeur INTEL Xeon quadri-cœurs hyperthreadés Nehalem X5550 [3] cadencés à 2,67 GHz. Elle dispose ainsi de 8 cœurs (16 threads) et de deux bancs mémoire de 24 Go, ainsi que de la technologie INTEL I/OAT. Elle est également dotée de trois cartes NVIDIA Quadro FX 5800 attachés aux différents nœuds NUMA par des liens PCIE de largeur 8x ou 16x comme illustré Figure A.8. Au cours des tests réseau effectués sur cette machine (Section 4.1.3), l'un de ces GPUs a été remplacé par une carte INFINIBAND QDR [4] (MELLANOX ConnectX MT26428).

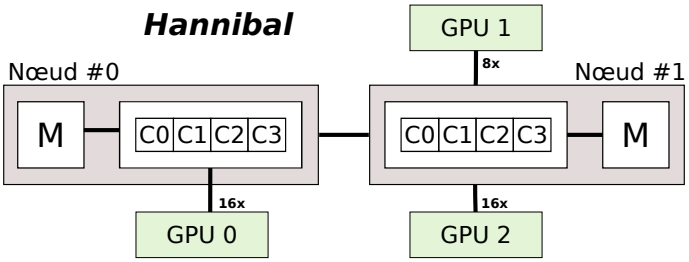


FIGURE A.8 – *Machine quadri-cœur* INTEL *Nehalem Hannibal.*

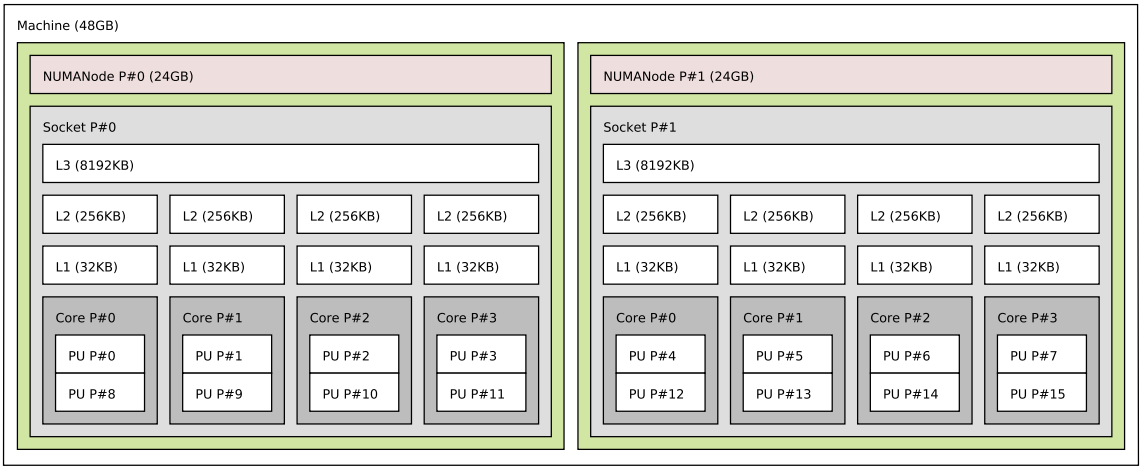


FIGURE A.9 – *Topologie de la machine Hannibal exposée par* `hwloc` .

3. Interconnectés par liens QUICKPATH INTERCONNECT (3,2 GHz)
4. *Quad Data Rate*

A.7 Bill

La machine *Bill* est une architecture SMP bi-processeur quadri-cœurs XEON Clovertown E5345, cadencés à 2,33 GHz. Elle est dotée de la technologie I/OAT et de caches L2 de 4 Mo partagés entre paires de cœurs comme illustré en Figure A.10.

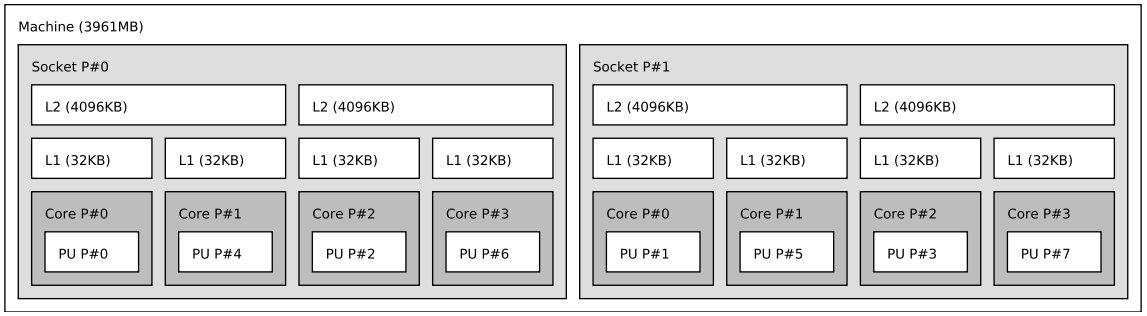


FIGURE A.10 – *Topologie de la machine Bill, bi-processeur Intel Xeon quadri-cœurs exposée par la bibliothèque* `hwloc` .

A.8 Idkonn

La plateforme *Idkonn* est une architecture SMP quadri-processeur INTEL Xeon hexa-cœurs Dunnington X7460 cadencés à 2,67 GHz. Cette plateforme a l'avantage d'offrir une hiérarchie de cache importante : sur chaque socket, un cœur dispose de son propre cache L1 de 32 ko, d'un cache L2 de 3 Mo partagé avec un cœur voisin, et d'un cache L3 de 16 Mo partagé entre les 6 cœurs (Figure A.11). Elle profite également de la technologie I/OAT.

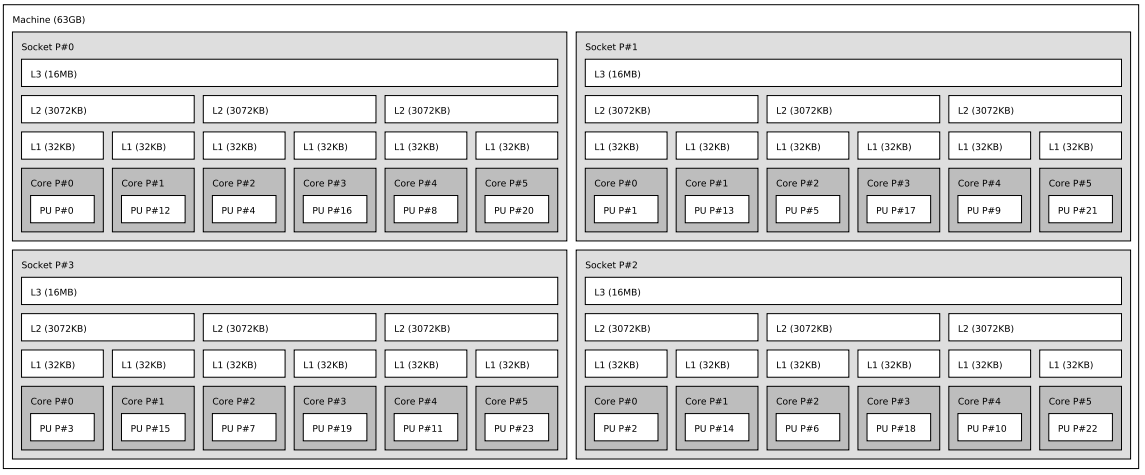


FIGURE A.11 – *Topologie de la machine Idkonn exposée par la bibliothèque* `hwloc` .

A.9 Bertha

La machine *Bertha* assemble en mémoire partagée avec cohérence de cache quatre serveur IBM x3950M2 contenant chacun du matériel équivalent à Idkonn A.8. Il s'agit donc d'une architecture à 4 nœuds NUMA contenant chacun quatre processeurs INTEL Xeon hexa-core Dunnington X7460 cadencés à 2,67 GHz. Elle offre donc un total de 96 cœurs, dispose de 4 bancs de 48 Go de RAM mais est cependant dépourvue de la technologie I/OAT.

Elle dispose de plus de 4 disques attachés aux nœuds 0, 2 et 3 comme illustré Figure A.12 sur lesquels ont été effectués les accès disques mentionnés en Section 4.1.4.

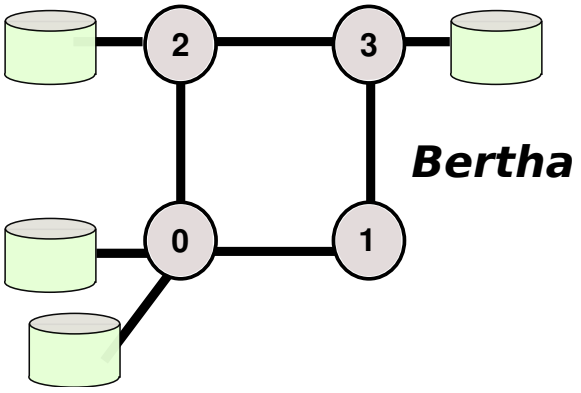


FIGURE A.12 – *Architecture 4 nœuds Bertha avec 4 disques connectés sur les nœuds 0, 2 et 3.*

FIGURE A.13 – *Architecture de la machine Bertha, quadri-processeur Intel Xeon hexa-cœurs Dunnington exposée par la bibliothèque* `hwloc` *.*

Annexe B

Performances des transferts de MPICH2-NEMESIS en mémoire partagée

Cette annexe présente une sélection de courbes complémentaires au Chapitre 6 permettant d'illustrer les performances de MPICH2-NEMESIS sur notre plateforme.

B.1 Performances des modes synchrone et asynchrone de KNEM

Le module noyau KNEM a été conçu pour supporter des transferts asynchrones, effectués soit à l'aide du support I/OAT des processeurs INTEL qui assure la copie en tâche de fond, soit à l'aide d'un thread noyau prenant en charge la copie mémoire.

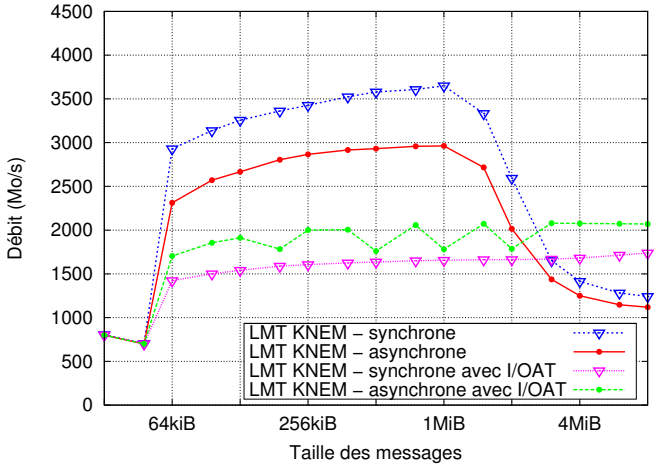


FIGURE B.1 – *Comparaison des performances du LMT* KNEM *en mode synchrone et asynchrone sur la machine Bill.*

La Figure B.1 présente les performances correspondantes obtenues sur l'architecture Bill (présentée en Annexe A.7). Dans le cas où la technologie I/OAT n'est pas mise à contribution, la compétition entre le processus MPI et le thread noyau pour l'utilisation du cœur se traduit par une perte notable de débit. Ce modèle pourrait toutefois bénéficier aux cas pour lesquels le nombre de processus est inférieur au nombre de cœurs, ou profiter de l'hyperthreading. Avec l'utilisation du déport de copie I/OAT, les performances bénéficient du modèle asynchrone puisque le travail est effectué en tâche de fond par du matériel dédié ne générant pas de compétition sur les cœurs. Par défaut, lorsque KNEM est activé avec le déport de copie I/OAT, le mode utilisé est ainsi basculé sur le mode asynchrone.

B.2 Comparaison des LMTs pour les transferts point-à-point

Les Figures B.2 et B.3 présentent le débit du test IMB *Pingpong* pour les différents LMTs sur la machine Bill (Annexe A.7). Pour ces tests, l'option *offcache* interdit la réutilisation des données contenues dans le cache. Les performances obtenues reflètent un avantage des stratégies de copie directe (vmsplice et KNEM) lorsqu'aucun cache n'est partagé, ainsi que le bénéfice évident du déport de copie I/OAT exempt de l'utilisation du cache dans tous les cas, pour cette architecture dont la bande passante mémoire est limitée.

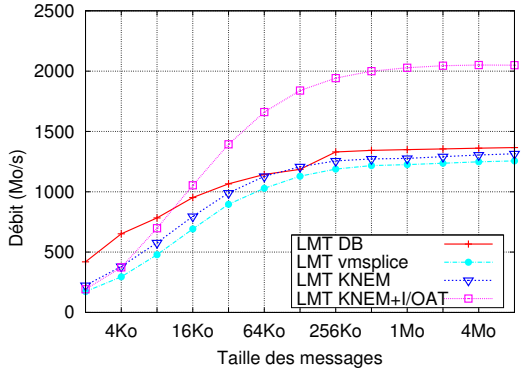


FIGURE B.2 – *Pingpong IMB entre 2 processus partageant un cache L2 de 4 Mo (machine Bill, offcache activé).*

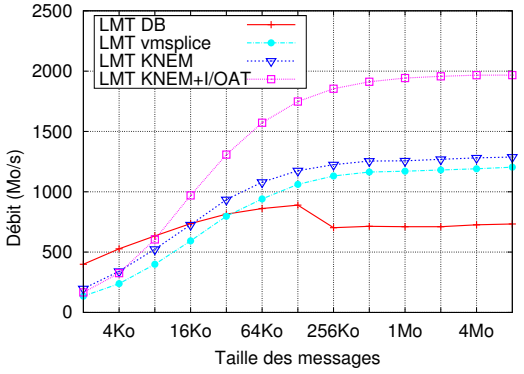


FIGURE B.3 – *Pingpong IMB entre 2 processus ne partageant aucun cache (machine Bill, offcache activé).*

Au contraire, sur la machine plus récente Hannibal, qui dispose de la technologie d'interconnexion mémoire QPI, les performances du déport de copie sont loin d'égaler celles des copies faites à l'initiative du processeur (Figure B.5 et B.4). La stratégie KNEM est cependant avantageuse lorsqu'aucun cache n'est partagé et rivalise avec le double-buffering dans le cas contraire.

Comme annoncé dans la Section 6.2.2, le LMT `vmsplice` apparaît comme une alternative à l'utilisation de KNEM en l'absence de cache commun aux cœurs sur lesquels sont exécutés les processus. Il offre cependant des performances inférieures au LMT KNEM quelque soit la configuration du matériel et des transferts (taille des messages et options).

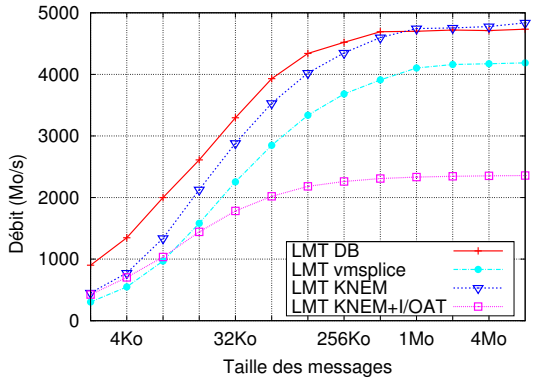


FIGURE B.4 – *Pingpong IMB entre 2 processus partageant un cache L3 de 8 Mo sur l'architecture Hannibal (offcache activé).*

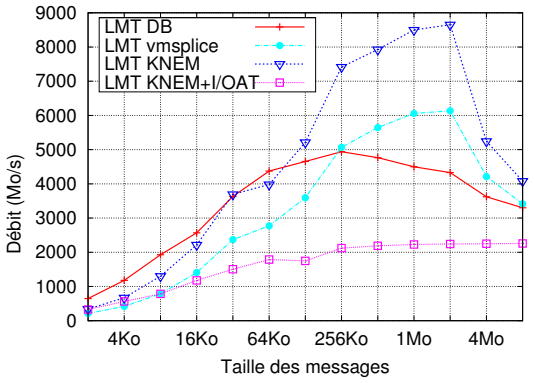


FIGURE B.5 – *Pingpong IMB entre 2 processus placés sur des nœuds distincts de la machine Hannibal.*

B.3 Performances des opérations collectives

Cette section présente quelques courbes de performances relatives aux opérations collectives effectuées grâce aux tests IMB sur différentes machines de notre plateforme de test.

B.3.1 Opérations *all-to-one*

Les Figures B.6 et B.7 présentent des exemples complémentaires de débits obtenus pour les opérations *reduce* et *allreduce*, que nous avons isolées comme présentant un comportement similaire. Comme mentionné en Section 6.2.3.2, ces opérations montrent un passage idéal à l'emploi du LMT KNEM autour de 128 ko pour le Reduce et légèrement inférieur (ici à 64 ko) pour l'opération *allreduce* qui génère davantage de tranferts. De plus, le déport de copie I/OAT ne présente pas de réel avantage bien qu'il profitait aux transferts point-à-point sur les hôtes ici cibles de nos tests.

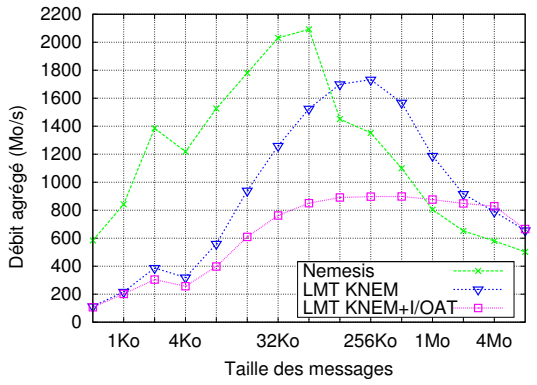


FIGURE B.6 – *Débits agrégés obtenus pour le test IMB Reduce sur la machine Bill (8 processus).*

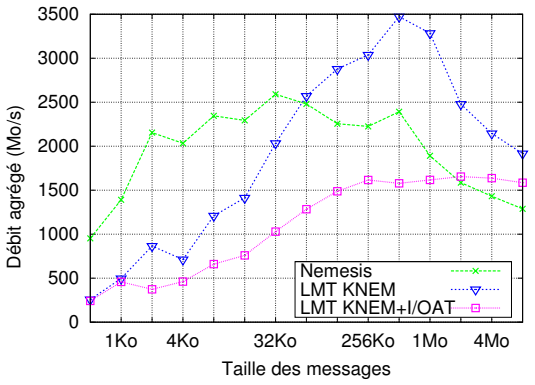


FIGURE B.7 – *Débits agrégés obtenus pour le test IMB Allreduce sur la machine Idkonn (16 processus).*

B.3.2 Opérations *one-to-all*

Nous présentons ici les résultats de deux tests de type *one-to-all*. La Figure B.8 illustre les performances de l'opération Scatter effectuée pour 8 processus sur la machine Bill. Nous observons ici un seuil de transition entre Nemesis et KNEM autour de 128 ko. Sur cette architecture, une amélioration de performance est liée à l'utilisation du déport de copie I/OAT tandis que cette stratégie s'avère inutile sur les autres machines de notre plateforme. La Figure B.9 présente un autre exemple de schéma de communication one-to-all : une opération de *broadcast* [1] effectuée entre 16 processus sur l'architecture Idkonn. Le seuil de passage à KNEM est ici légèrement inférieur (64 ko) et la stratégie KNEM+I/OAT montre des performances médiocres que nous pensons dues à une forte contention alors que le moteur DMA doit traiter les transferts pour 16 processus.

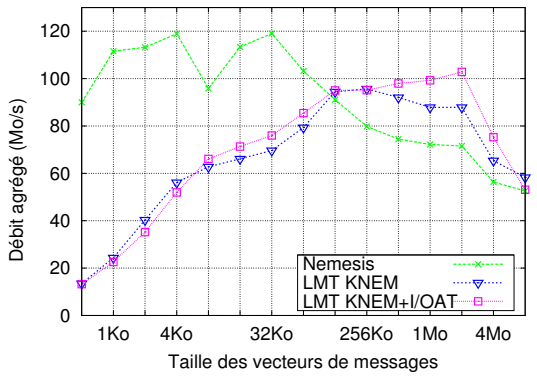


FIGURE B.8 – *Débits agrégés obtenus pour le test IMB Scatter sur la machine Bill (8 processus).*

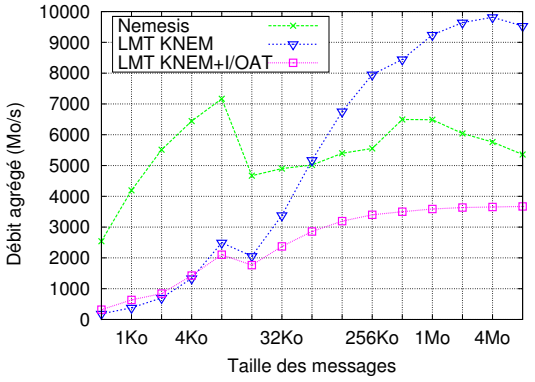


FIGURE B.9 – *Débits agrégés obtenus pour le test IMB Bcast sur la machine Hannibal (8 processus).*

B.3.3 Opérations *all-to-all*

Les courbes présentées en Figures B.10 et B.11 exposent le comportement observé pour les opérations *allgather*. Comme pour les tests de *alltoall*, les seuils de passage de Nemesis à KNEM sont d'autant plus faible que le nombre de processus est important (4 ko pour 8 processus sur la machine Bill et 2 ko pour 16 processus sur Idkonn). Encore une fois, seule l'architecture Bill semble bénéficier du déport de copie I/OAT.

B.4 Performances applicatives des différents LMTs pour les tests NAS

La Table B.1 présente les temps d'exécution de différents tests de la suite de tests parallèles NAS [29] sur notre machine Bill. La plupart d'entre eux n'utilisent que très peu de larges mes-

1. Contrairement à l'opération de scatter, les données ne sont pas divisées en plusieurs segments transmis aux différents processus, mais envoyées à chaque processus dans leur intégralité générant donc des tailles de messages plus grandes.

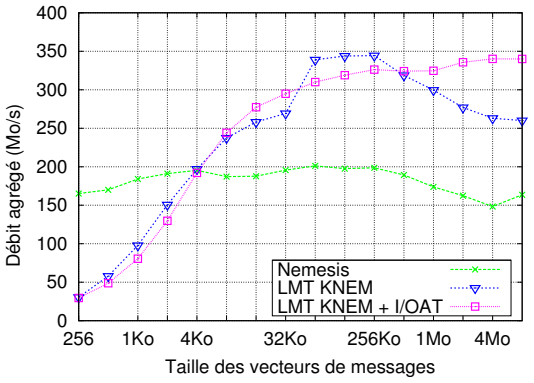


FIGURE B.10 – *Débits agrégés obtenus pour le test IMB Allgather sur la machine Bill (8 processus).*

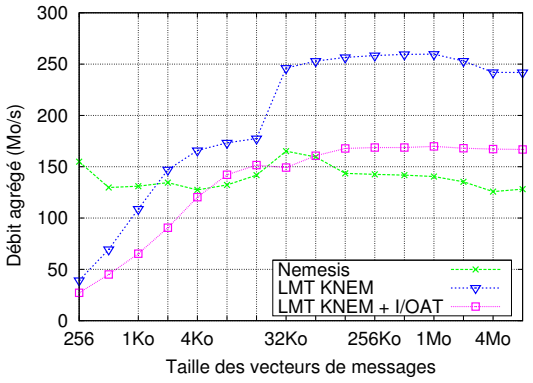


FIGURE B.11 – *Débits agrégés obtenus pour le test IMB Allgather sur la machine Idkonn (16 processus).*

sages, et ne montrent ainsi que de légères variations de performance. Le test NAS IS, connu pour son utilisation de très grands messages montre lui un gain de 25% sur son temps d'exécution grâce à l'utilisation de KNEM avec I/OAT, et le test FT un gain de 10%.

Test NAS	LMT double-buffering	LMT `vmsplice`	KNEM copie noyau	KNEM I/OAT	Amélioration
bt.B.4	454.3 s	452.1 s	453.6 s	452.3 s	+ 0.4%
cg.B.8	60.26 s	61.87 s	60.72 s	61.59 s	- 2.2%
ep.B.4	30.45 s	30.94 s	32.40 s	30.72 s	- 0.9%
ft.B.8	39.25 s	37.00 s	36.40 s	35.50 s	**+ 10.6%**
is.B.8	2.34 s	1.95 s	1.92 s	1.86 s	**+ 25.8%**
lu.B.8	85.83 s	87.45 s	86.09 s	88.32 s	- 2.9%
mg.B.8	7.81 s	ND [2]	7.89 s	7.98 s	- 2.1%
sp.B.8	302.0 s	311.4 s	298.9 s	299.4 s	+ 0.9%

TABLE B.1 – *Temps d'exécution de différents tests parallèles de la suite NAS sur notre architectures Bill.*

Pour expliquer le bénéfice apporté au test IS, la Table B.2 présente le nombre de défauts de cache [3] produits, mesuré à l'aide de PAPI [17]. Ces résultats montrent une corrélation entre le temps d'exécution de ce test et le nombre de défauts de cache. En utilisant une copie directe avec KNEM, ou avec KNEM et I/OAT, la pollution de cache est réduite. Le nombre de défauts de cache diminue ainsi, d'où une réduction du temps d'exécution.

L'observation des comportements point-à-point (*Pingpong*) et collectif (*Alltoall*), confirme en effet que KNEM évite un nombre de défauts de cache significatif pour le transfert de grands messages,

2. Cette absence est due à un bug connu mais encore non résolu dans NEMESIS.
3. Le pourcentage de défaut de cache n'est pas présenté ici car les stratégies comparées ont un nombre de requêtes aux caches dépendant fortement de leur implémentation et donc non comparables entre eux.

	LMT double-buffering	LMT `vmsplice`	KNEM copie noyau	KNEM I/OAT
Pingpong 64 ko	91	166	52	92
Pingpong 4 Mo	45k	17k	14k	3.7k
Alltoall 64 ko	2783	1266	582	833
Alltoall 4 Mo	624k	124k	262k	131k
is.B.8	11.25M	9.41M	9.50M	8.92M

TABLE B.2 – *Défauts de cache L2 pendant l'exécution de différents tests. IS et Alltoall utilisent les 8 cœurs de la machine. Lors des Pingpong les processus sont placés sur des puces différentes.*

tandis que l'implémentation de la copie *double-buffering* de NEMESIS ne reste compétitive que pour de petits messages. Comme attendu, les performances exhibées par la stratégie vmsplice positionne ce LMT comme une alternative intéressante lorsque le module KNEM ne peut être chargé.

Bibliographie

[1] DASSAULT AVIATION AND DASSAULT FALCON JET CORPORATION. « Dassault Falcon ». http://www.dassaultfalcon.com/.

[2] GOMES (R.), LEVISON (H. F.), TSIGANIS (K.) et MORBIDELLI (A.), « Origin of the cataclysmic late heavy bombardment period of the terrestrial planets », *Nature*, vol. 435, n° 7041, mai 2005, p. 466–469.

[3] MORBIDELLI (A.), LEVISON (H. F.), TSIGANIS (K.) et GOMES (R.), « Chaotic capture of Jupiter's trojan asteroids in the early solar system », *Nature*, vol. 435, n° 7041, mai 2005, p. 462–465.

[4] TSIGANIS (K.), GOMES (R.), MORBIDELLI (A.) et LEVISON (H. F.), « Origin of the orbital architecture of the giant planets of the solar system », *Nature*, vol. 435, n° 7041, mai 2005, p. 459–461.

[5] TOP500. « Top500 supercomputing sites ». http://www.top500.org/.

[6] « Grid'5000 ». https://www.grid5000.fr/mediawiki/index.php/Grid5000:Home.

[7] ADVANCED MICRO DEVICES, INC.. « The AMD Fusion Family of APUs ». http://sites.amd.com/us/fusion/apu/Pages/fusion.aspx.

[8] « IOzone Filesystem Benchmark ». http://www.iozone.org/docs/IOzone_msword_98.pdf.

[9] KLEEN (A.). « A NUMA API for LINUX ». Novell, Technical Linux Whitepaper, avril 2005.

[10] « Portable Linux Processor Affinity (PLPA) ». http://www.open-mpi.org/projects/plpa.

[11] HENNESSY (J. L.) et PATTERSON (D. A.), *Computer Architecture : A Quantitative Approach*. Morgan Kaufman, 3e édition, 2003.

[12] MOSLEY (L. E.). « Power delivery challenges for multi-core microprocessors », 2008. http://ecadigitallibrary.com/pdf/CARTSUSA08/2_0a%20Mosley-Intel.pdf.

[13] KALEMKARIAN (Y.). « MPI and multi-core architectures ». http://www.cse.scitech.ac.uk/disco/mew17/talks/Yann_multicore_architectures__061206.pdf.

[14] MATTSON (T. G.), RIEPEN (M.), LEHNIG (T.) *et al.*, « The 48-core scc processor : the programmer's view », dans *Proceedings of the 2010 ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, coll. « SC '10 », p. 1–11, Washington, DC, USA, 2010. IEEE Computer Society.

[15] INTEL. « Tera-scale Computing Research Program ». `http://techresearch.intel.com/ResearchAreaDetails.aspx?Id=27#Vision`.

[16] « The splice() weekly news », avril 2006. `http://lwn.net/Articles/181169/`.

[17] BROWNE (S.), DONGARRA (J.), GARNER (N.) *et al.*, « A Portable Programming Interface for Performance Evaluation on Modern Processors », *The International Journal of High Performance Computing Applications*, vol. 14, n° 3, 2000, p. 189–204.

[18] YANG (R.), ANTONY (J.), JANES (P. P.) et RENDELL (A. P.), « Memory and Thread Placement Effects as a Function of Cache Usage : A Study of the Gaussian Chemistry Code on the SunFire X4600 M2 », dans *Proceedings of the International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN 2008)*, p. 31–36, 2008.

[19] FRIGO (M.), LEISERSON (C. E.), PROKOP (H.) et RAMACHANDRAN (S.), « Cache-Oblivious Algorithms », dans *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, p. 285, New York City, NY, octobre 1999. IEEE Computer Society.

[20] FEDOROVA (M. S. A.), « Cache-fair thread scheduling for multicore processors ». Rapport technique, Division of Engineering and Applied Sciences, Harvard University, octobre 2006.

[21] FEDOROVA (A.), *Operating System Scheduling for Chip Multithreaded Processors*. Thèse de doctorat, Université de Harvard, Cambridge, MA, septembre 2006.

[22] JUCKEL (G.), MÜLLER (M. S.), NAGEL (W. E.) *et al.*, « Accessing Data on SGI Altix : An Experience with Reality », dans *The 4th Workshop on Memory Performance Issues (WMPI-2006)*, 2006.

[23] MCDOUGALL (R.) et MAURO (J.), *Solaris^{TM} Internals : Solaris 10 and OpenSolaris Kernel Architecture*. Prentice Hall, 2e édition, juillet 2006.

[24] STERLING (T.), SAVARESE (D.), BECKER (D. J.) *et al.*, « BEOWULF : A parallel workstation for scientific computation », dans *Proceedings of the 24th International Conference on Parallel Processing*, p. I :11–14, Oconomowoc, WI, 1995.

[25] ANDERSON (T. E.), CULLER (D. E.) et PATTERSON (D. A.), « A Case for NOW (Networks of Workstations », *IEEE Micro*, vol. 15, n° 1, février 1995, p. 54–64.

[26] SONG (F.), MOORE (S.) et DONGARRA (J.), « L2 Cache Modeling for Scientific Applications on Chip Multi-Processors », dans *Proceedings of International Conference on Parallel Processing (ICPP-2007)*, p. 51, septembre 2007.

[27] SONG (F.), MOORE (S.) et DONGARRA (J.), « Modeling of L2 cache behavior for thread-parallel scientific programs on Chip Multi-Processors ». Rapport technique, UT Computer Science, 2006.

[28] AUGONNET (C.), THIBAULT (S.) et NAMYST (R.), « StarPU : a Runtime System for Scheduling Tasks over Accelerator-Based Multicore Machines ». Rapport de recherche n° RR-7240, INRIA, mars 2010.

[29] BAILEY (D. H.), BARSZCZ (E.), BARTON (J. T.) *et al.*, « The NAS Parallel Benchmarks », *The International Journal of Supercomputer Applications*, vol. 5, n° 3, 1991, p. 63–73.

[30] MCCALPIN (J. D.), « STREAM : Sustainable Memory Bandwidth in High Performance Computers ». Rapport technique, University of Virginia, Charlottesville, Virginia, 1991-2007. A continually updated technical report. http ://www.cs.virginia.edu/stream/.

[31] « Intel MPI Benchmarks ». `http://software.intel.com/en-us/articles/intel-mpi-benchmarks/`.

[32] « LMbench - Tools for Performance Analysis ». `http://www.bitmover.com/lmbench/`.

[33] BROQUEDIS (F.), CLET-ORTEGA (J.), MOREAUD (S.) *et al.*, « hwloc : a Generic Framework for Managing Hardware Affinities in HPC Applications », dans *Proceedings of the 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP2010)*, p. 180–186, Pise, Italie, février 2010. IEEE Computer Society Press.

[34] BRECHT (T.), « On the Importance of Parallel Application Placement in NUMA Multiprocessors », dans *Proceedings of the 4th Symposium on Experiences with Distributed and Multiprocessor Systems (SEDMS IV)*, San Diego, CA, septembre 1993.

[35] ROBERTSON (N.) et RENDELL (A.), « OpenMP and NUMA Architectures I : Investigating Memory Placement on the SGI Origin 3000 », dans VERLAG (S.), éditeur, *3rd International Conference on Computational Science*, vol. 2660 (coll. *Lect. Notes in Comp. Science*), p. 648–656, 2003.

[36] CHAI (L.), GAO (Q.) et PANDA (D. K.), « Understanding the Impact of Multi-Core Architecture in Cluster Computing : A Case Study with Intel Dual-Core System », dans *Proceedings of the International Symposium on Cluster Computing and the Grid (CCGrid)*, Rio de Janeiro, Brézil, mai 2007.

[37] BADIA (R. M.), PEREZ (J. M.), AYGUADÉ (E.) et LABARTA (J.), « Impact of the memory hierarchy on shared memory architectures in multicore programming models », dans *Proceedings of the 2009 17th Euromicro International Conference on Parallel, Distributed and Network-based Processing*, p. 437–445, Washington, DC, USA, février 2009. IEEE Computer Society.

[38] NARAYANASWAMY (G.), BALAJI (P.) et FENG (W.), « Impact of Network Sharing in Multicore Architectures », dans *Proceedings of the 17th International Conference on Computer Communication and Networks (ICCCN'08)*, St. Thomas, U.S. Virgin Islands, août 2008. IEEE Computer Society.

[39] « HyperTransport I/O Link Specification ». `http://www.hypertransport.org`.

[40] ADVANCED MICRO DEVICES, INC.. « HyperTransport Technology I/O Link, A High-Bandwidth I/O Architecture », juillet 2001.

[41] KELTCHER (C. N.), MCGRATH (K. J.), AHMED (A.) et CONWAY (P.), « The AMD Opteron Processor for Multiprocessor Servers », *IEEE Micro*, vol. 23, n° 2, mars 2003, p. 66–76.

[42] ANTONY (J.), JANES (P. P.) et RENDELL (A. P.), « Exploring Thread and Memory Placement on NUMA Architectures : Solaris and Linux, UltraSPARC/FirePlane and Opteron/HyperTransport », dans *Proceedings of the International Conference on High Performance Computing (HiPC)*, Bangalore, Inde, décembre 2006.

[43] INTEL CORPORATION. « First the Tick, Now the Tock : Next Generation Intel Microarchitecture (Nehalem) ». White paper, avril 2008.

[44] INTEL CORPORATION. « Intel QuickPath Architecture : A new system architecture for unleashing the performance of future generations of Intel multi-core microprocessors ». White paper, 2008.

[45] BARKER (K. J.), DAVIS (K.), HOISIE (A.) *et al.*, « A Performance Evaluation of the Nehalem Quad-Core Processor for Scientific Computing », dans *Parallel Processing Letters*, vol. 18, p. 453–469, 2008.

[46] BEECROFT (J.), ADDISON (D.), PETRINI (F.) et MCLAREN (M.), « Quadrics QsNet II : A network for Supercomputing Applications », dans *The Proceedings of Hot Chips 15*, Stanford University, Palo Alto, CA, août 2003.

[47] PETRINI (F.), CHUN FENG (W.), HOISIE (A.) *et al.*, « The Quadrics Network : High-Performance Clustering Technology », *IEEE Micro*, vol. 22, nº 1, 2002, p. 46–57.

[48] « InfiniBand architecture specifications ». InfiniBand Trade Association, 2001. `http://www.infinibandta.org`.

[49] « OpenFabrics Alliance ». `http://www.openfabrics.org`.

[50] KOOP (M.), HUANG (W.), GOPALAKRISHNAN (K.) et PANDA (D. K.), « Performance Analysis and Evaluation of PCIe 2.0 and Quad-Data Rate InfiniBand », dans *16th Int'l Symposium on Hot Interconnects (HotI16)*, Palo Alto, CA, août 2008. IEEE Computer Society.

[51] « Myricom Inc. ». `http://www.myri.com`.

[52] BODEN (N. J.), COHEN (D.), FELDERMAN (R. E.) *et al.*, « Myrinet : A Gigabit-per-Second Local Area Network », *IEEE Micro*, vol. 15, nº 1, 1995, p. 29–36.

[53] Myricom, Inc., *GM : A message-passing system for Myrinet networks*, 2003. `http://www.myri.com/scs/GM/doc/html/`.

[54] Myricom, Inc., *Myrinet Express (MX) : A High Performance, Low-Level, Message-Passing Interface for Myrinet*, 2006. `http://www.myri.com/scs/MX/doc/mx.pdf`.

[55] DUBNICKI (C.), BILAS (A.), LI (K.) et PHILBIN (J.), « Design and Implementation of Virtual Memory-Mapped Communication on Myrinet », dans *Proceedings of the 11th International Symposium on Parallel Processing (IPPS)*, p. 388, 1997.

[56] LAURIA (M.), PAKIN (S.) et CHIEN (A. A.), « Efficient layering for high speed communication : Fast Messages 2.x », dans *Proceedings of The Seventh International Symposium on High Performance Distributed Computing*, p. 10–20, jul 1998.

[57] PRYLLI (L.) et TOURANCHEAU (B.), « BIP : A new protocol designed for high performance networking on Myrinet », dans ROLIM (J.), éditeur, *Parallel and Distributed Processing*, vol. 1388 (coll. *Lecture Notes in Computer Science*), p. 472–485. Springer Berlin / Heidelberg, 1998.

[58] GEOFFRAY (P.), PRYLLI (L.) et TOURANCHEAU (B.), « BIP-SMP : High Performance Message Passing over a Cluster of Commodity SMPs », dans *Proceedings of the 13th International Conference on Supercomputing (ICS'99)*, Portland, Oregon, USA, novembre 1999.

[59] CONWAY (P.), KALYANASUNDHARAM (N.), DONLEY (G.) *et al.*, « Cache Hierarchy and Memory Subsystem of the AMD Opteron Processor », *IEEE Micro*, vol. 30, nº 2, mars 2010, p. 16 –29.

[60] ALMÁSI (G.), HEIDELBERGER (P.), ARCHER (C. J.) *et al.*, « Optimization of MPI collective communication on BlueGene/L systems », dans *Proceedings of the 19th International Conference on Supercomputing (ICS'05)*, p. 253–262, Cambridge, MA, 2005. ACM.

[61] MERCIER (G.) et JEANNOT (E.), « Improving MPI Applications Performance on Multicore Clusters with Rank Reordering. », dans *Proceedings of the 16th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface (EuroPVM/MPI 2011)*, Santorini, Grèce, septembre 2011.

[62] « MPICH2 : High-performance and Widely Portable MPI ». `http://www.mcs.anl.gov/research/projects/mpich2/`.

[63] BUNTINAS (D.), MERCIER (G.) et GROPP (W.), « Implementation and Shared-Memory Evaluation of MPICH2 over the Nemesis Communication Subsystem », dans *Proceedings of the 13th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface (EuroPVM/MPI 2006)*, Bonn, Allemagne, septembre 2006.

[64] BUNTINAS (D.), MERCIER (G.) et GROPP (W.), « Design and Evaluation of Nemesis, a Scalable Low-Latency Message-Passing Communication Subsystem », dans TURNER (S. J.), LEE (B. S.) et CAI (W.), éditeurs, *Proceedings of the 6th International Symposium on Cluster Computing and the Grid (CCGRID '06)*, p. 521–530, Washington, DC, USA, mai 2006. IEEE Computer Society.

[65] BUNTINAS (D.), MERCIER (G.) et GROPP (W.), « Implementation and Evaluation of Shared-Memory Communication and Synchronization Operations in MPICH2 using the Nemesis Communication Subsystem », *Parallel Computing, Selected Papers from EuroPVM/MPI 2006*, vol. 33, n° 9, septembre 2007, p. 634–644.

[66] BUNTINAS (D.), GOGLIN (B.), GOODELL (D.) *et al.*, « Cache-Efficient, Intranode Large-Message MPI Communication with MPICH2-Nemesis », dans *Proceedings of the 38th International Conference on Parallel Processing (ICPP-2009)*, p. 462–469, Vienne, Autriche, septembre 2009. IEEE Computer Society Press.

[67] MERCIER (G.), TRAHAY (F.), BUNTINAS (D.) et BRUNET (É.), « NewMadeleine : An Efficient Support for High-Performance Networks in MPICH2 », dans *Proceedings of 23rd International Parallel and Distributed Processing Symposium (IPDPS'09)*, Rome, Italie, mai 2009. IEEE Computer Society.

[68] GABRIEL (E.), FAGG (G. E.), BOSILCA (G.) *et al.*, « Open MPI : Goals, Concept, and Design of a Next Generation MPI Implementation », dans *Proceedings of the 11th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface (EuroPVM/MPI 2004)*, p. 97–104, Budapest, Hongrie, septembre 2004.

[69] GRAHAM (R. L.), WOODALL (T. S.) et SQUYRES (J. M.), « Open MPI : A Flexible High Performance MPI », dans *Proceedings, 6th Annual International Conference on Parallel Processing and Applied Mathematics*, Poznan, Pologne, septembre 2005.

[70] SQUYRES (J. M.) et LUMSDAINE (A.), « The Component Architecture of Open MPI : Enabling Third-Party Collective Algorithms », dans GETOV (V.) et KIELMANN (T.), éditeurs, *Proceedings of the 18th International Conference on Supercomputing (ISC'04), Workshop on Component Models and Systems for Grid Applications*, p. 167–185, St. Malo, France, juillet 2004. Springer.

[71] FAGG (G.), BOSILCA (G.), PJEŠIVAC-GRBOVIĆ (J.) *et al.*, « Tuned : An Open MPI Collective Communications Component », dans KACSUK (P.), FAHRINGER (T.) et NÉMETH (Z.), éditeurs, *Distributed and Parallel Systems*, p. 65–72. Springer, 2007.

[72] MA (T.), BOSILCA (G.), BOUTEILLER (A.) et DONGARRA (J. J.), « Locality and Topology aware Intra-node Communication Among Multicore CPUs », dans *Proceedings of the 17th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface (EuroPVM/MPI 2010)*, coll. « Lecture Notes in Computer Science », Stuttgart, Allemagne, septembre 2010. Springer.

[73] GRAHAM (R. L.), SHIPMAN (G. M.), BARRETT (B. W.) *et al.*, « Open MPI : A High-Performance, Heterogeneous MPI », dans *Proceedings of the 5th International Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks (HeteroPar'2009)*, Barcelone, Espagne, septembre 2006.

[74] PELLEGRINI (S.), WANG (J.), FAHRINGER (T.) et MORITSCH (H.), « Optimizing MPI Runtime Parameter Settings by Using Machine Learning », dans *Proceedings of the 16th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface (EuroPVM/MPI 2009)*, vol. 5759 (coll. *Lecture Notes in Computer Science*), p. 196–206, Espoo, Finlande, septembre 2009. Springer.

[75] FARAJ (A.) et YUAN (X.), « Automatic generation and tuning of MPI collective communication routines », dans *Proceedings of the 19th International Conference on Supercomputing (ICS'05),*, p. 393–402, Cambridge, MA, 2005. ACM.

[76] FARAJ (A.), YUAN (X.) et LOWENTHAL (D.), « STAR-MPI : self tuned adaptive routines for MPI collective operations », dans *Proceedings of the 20th International Conference on Supercomputing (ICS'06)*, p. 199–208, Cairns, Queensland, Australie, 2006. ACM.

[77] CHAARAWI (M.), SQUYRES (J. M.), GABRIEL (E.) et FEKI (S.), « A Tool for Optimizing Runtime Parameters of Open MPI », dans *Proceedings of the 15th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface (EuroPVM/MPI 2008)*, p. 210–217, Dublin, Irelande, 2008. Springer-Verlag.

[78] NETWORK-BASED COMPUTING LAB, THE OHIO STATE UNIVERSITY. « MVAPICH : MPI over InfiniBand and iWARP ». `http://mvapich.cse.ohio-state.edu/`.

[79] JIN (H.-W.), SUR (S.), CHAI (L.) et PANDA (D. K.), « LiMIC : Support for High-Performance MPI Intra-Node Communication on Linux Cluster », dans *Proceedings of the International Conference on Parallel Processing (ICPP-2005)*, Oslo, Norvège, juin 2005. IEEE Computer Society.

[80] LIU (J.), WU (J.) et PANDA (D. K.), « High Performance RDMA-Based MPI Implementation over InfiniBand », *International Journal of Parallel Programming*, vol. 32, n° 3, juin 2004.

[81] CHAI (L.), HARTONO (A.) et PANDA (D. K.), « Designing High Performance and Scalable MPI Intra-node Communication Support for Clusters », dans *Proceedings of the International Conference on Cluster Computing (Cluster'06)*, Barcelone, Espagne, septembre 2006. IEEE Computer Society.

[82] JIN (H.-W.), SUR (S.), CHAI (L.) et PANDA (D. K.), « Lightweight Kernel-Level Primitives for High-Performance MPI Intra-Node Communication over Multi-Core Systems », dans *Proceedings of the International Conference on Cluster Computing (Cluster'07)*, Austin, TX, septembre 2007. IEEE Computer Society.

[83] CHAI (L.), LAI (P.), JIN (H.-W.) et PANDA (D. K.), « Designing An Efficient Kernel-level and User-level Hybrid Approach for MPI Intra-node Communication on Multi-core Sys-

tems », dans *Proceedings of the International Conference on Parallel Processing (ICPP-2008)*, Portland, Oregon, septembre 2008. IEEE Computer Society Press.

[84] NIKOLOPOULOS (D. S.), PAPATHEODOROU (T. S.), POLYCHRONOPOULOS (C. D.) *et al.*, « User-Level Dynamic Page Migration for Multiprogrammed Shared-Memory Multiprocessors », dans *Proceedings of the International Conference on Parallel Processing (ICPP-2000)*, p. 95–103. IEEE Computer Society, septembre 2000.

[85] GOGLIN (B.) et FURMENTO (N.), « Enabling High-Performance Memory-Migration in Linux for Multithreaded Applications », dans *Proceedings of the Workshop on Multithreaded Architectures and Applications (MTAAP'09), tenu conjointement avec IPDPS'09*, Rome, Italie, mai 2009. IEEE Computer Society.

[86] MARATHE (J.) et MUELLER (F.), « Hardware Profile-guided Automatic Page Placement for ccNUMA systems », dans *Proceedings of the 6th Symposium on Principles and Practice of Parallel Programming (PPoPP)*, Manhattan, NY, mars 2006.

[87] LAMETER (C.), « Local and Remote Memory : Memory in a Linux/NUMA System », dans *Linux Symposium (OLS2006)*, Ottawa, Canada, juillet 2006.

[88] LÖF (H.) et HOLMGREN (S.), « Affinity-on-next-touch : Increasing the Performance of an Industrial PDE Solver on a cc-NUMA System », dans *Proceedings of the 19th International Conference on Supercomputing (ICS'05)*, p. 387–392, Cambridge, MA, novembre 2005.

[89] TERBOVEN (C.), AN MEY (D.), SCHMIDL (D.) *et al.*, « Data and Thread Affinity in OpenMP Programs », dans *MAW '08 : Proceedings of the 2008 workshop on Memory access on future processors*, p. 377–384, Ischia, Italy, 2008. ACM.

[90] WHITNEY (S.), MCCALPIN (J.), BITAR (N.) *et al.*, « The SGI Origin Software Environment and Application Performance », dans *Proceedings of COMPCON 97*, p. 165–170, San Jose, CA, 1997. IEEE Computer Society.

[91] STECKERMEIER (M.) et BELLOSA (F.), « Using Locality Information in Userlevel Scheduling ». Rapport technique n° TR-95-14, University of Erlangen-Nürnberg – Computer Science Department – Operating Systems – IMMD IV, Allemagne, décembre 1995.

[92] LAI (G.-J.) et CHEN (C.), « A New Scheduling Strategy for NUMA Multiprocessor Systems », dans *Proceedings of the International Conference on Parallel and Distributed Systems (ICPADS '96)*, p. 222–229, Tokyo, Japon, juin 1996.

[93] NARLIKAR (G. J.), « Scheduling Threads for Low Space Requirement and Good Locality », *Theory of Computing Systems*, vol. 35, n° 2, janvier 2002, p. 151–187.

[94] THIBAULT (S.), « A Flexible Thread Scheduler for Hierarchical Multiprocessor Machines », dans *Proceedings of the Second International Workshop on Operating Systems, Programming Environments and Management Tools for High-Performance Computing on Clusters (COSET-2)*, Cambridge, MA, juin 2005.

[95] CHANDRA (R.), DEVINE (S.), VERGHESE (B.) *et al.*, « Scheduling and Page Migration for Multiprocessor Compute Servers », dans *Proceedings of the 6th international conference on Architectural support for programming languages and operating systems table of contents (ASPLOS-VI)*, p. 12–24, San Jose, CA, 1994.

[96] MARKATOS (E.) et LEBLANC (T.), « Using processor affinity in loop scheduling on shared-memory multiprocessors », *Parallel and Distributed Systems*, vol. 5, n° 4, avril 1994, p. 379–400.

[97] LI (H.), TANDRI (S.), STUMM (M.) et SEVCIK (K. C.), « Locality and Loop Scheduling on NUMA Multiprocessors », dans *Proceedings of the International Conference on Parallel Processing (ICPP-1993)*, vol. 2, p. 140–127, août 1993.

[98] WANG (Y.-M.), WANG (H.-H.) et CHANG (R.-C.), « Hierarchical loop scheduling for clustered NUMA machines », *Journal of Systems and Software*, vol. 55, décembre 2000, p. 33–44.

[99] ALAM (S. R.), BARRETT (R. F.), KUEHN (J. A.) *et al.*, « Characterization of Scientific Workloads on Systems with Multi-Core Processors », dans *Proceedings of the International Symposium on Workload Characterization (IISWC)*, San Jose, CA, 2006. IEEE Computer Society.

[100] SONG (F.), MOORE (S.) et DONGARRA (J.), « Feedback-Directed Thread Scheduling with Memory Considerations », dans *Proceedings of the 16th International Symposium on High-Performance Distributed Computing (HPDC07)*, Monterey Bay, CA, juin 2007. IEEE Computer Society.

[101] SONG (F.), MOORE (S.) et DONGARRA (J.), « Analytical Modeling and Optimization for Affinity Based Thread Scheduling on Multicore Systems », dans *Proceedings of the International Conference on Cluster Computing (Cluster'09)*, Nouvelle Orleans, LA, août 2009. IEEE Computer Society.

[102] THIBAULT (S.), *Ordonnancement de processus légers sur architectures multiprocesseurs hiérarchiques : BubbleSched, une approche exploitant la structure du parallélisme des applications*. Thèse de doctorat, Université Bordeaux 1, décembre 2007.

[103] THIBAULT (S.), NAMYST (R.) et WACRENIER (P.-A.), « Building Portable Thread Schedulers for Hierarchical Multiprocessors : the BubbleSched Framework », dans *Proceedings of the 13th European Conference on Parallel and Distributed Computing (Euro-Par'07)*, Rennes, France, août 2007. ACM.

[104] JEULAND (S.), « Ordonnancement de threads dirigé par la mémoire sur architecture NUMA ». Mémoire de master recherche, Université Bordeaux 1, septembre 2007.

[105] BROQUEDIS (F.), *De l'exécution d'applications scientifiques OpenMP sur architectures hiérarchiques*. Thèse de doctorat, Université Bordeaux 1, décembre 2010.

[106] BROQUEDIS (F.), FURMENTO (N.), GOGLIN (B.) *et al.*, « ForestGOMP : an efficient OpenMP environment for NUMA architectures », *International Journal on Parallel Programming, Special Issue on OpenMP ; Guest Editors : Matthias S. Müller and Eduard Ayguadé*, vol. 38, n° 5, 2010, p. 418–439.

[107] AUMAGE (O.), BRUNET (E.), MERCIER (G.) et NAMYST (R.), « High-Performance Multi-Rail Support with the NewMadeleine Communication Library », dans *Proceedings of the 16th International Heterogeneity in Computing Workshop (HCW'07), tenu conjointement avec IPDPS'07*, Long Beach, CA, mars 2007.

[108] AUMAGE (O.), BRUNET (E.), FURMENTO (N.) et NAMYST (R.), « NewMadeleine : a Fast Communication Scheduling Engine for High Performance Networks », dans *Proceedings of the 7th Workshop on Communication Architecture for Clusters (CAC'07), tenu conjointement avec IPDPS'07*, Long Beach, CA, mars 2007.

[109] « OpenMP : The OpenMP API specification for parallel programming ». `http://openmp.org`.

[110] FORUM (M. P. I.), « MPI : A message-passing interface standard ». Rapport technique n° UT-CS-94-230, 1994.

[111] « Message Passing Interface ». `http://www-unix.mcs.anl.gov/mpi/`.

[112] « MPI-2 : Extensions to the Message-Passing Interface ». `http://www.mpi-forum.org/docs/mpi-20-html/mpi2-report.html`.

[113] MESSAGE PASSING INTERFACE FORUM. « MPI : A Message-Passing Interface Standard, Version 2.1 », June 2008. `http://www.mpi-forum.org/docs/mpi21-report.pdf`.

[114] CIACCIO (G.) et CHIOLA (G.), « GAMMA and MPI/GAMMA on Gigabit Ethernet », dans *Proceedings of 7th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface (EuroPVM/MPI 2000)*, p. 129–136, Balatonfured, Hongrie, septembre 2000.

[115] GOGLIN (B.), « Design and Implementation of Open-MX : High-Performance Message Passing over generic Ethernet hardware », dans *Proceedings of the 8th Workshop on Communication Architecture for Clusters (CAC'08), tenu conjointement avec IPDPS'08*, Miami, FL, avril 2008. IEEE Computer Society.

[116] GOGLIN (B.), « High-Performance Message Passing over generic Ethernet Hardware with Open-MX », *Journal of Parallel Computing*, vol. 37, n° 2, février 2011, p. 85–100. Open-MX.

[117] MA (T.), BOSILCA (G.), BOUTEILLER (A.) *et al.*, « Kernel Assisted Collective Intra-node MPI Communication Among Multi-core and Many-core CPUs », dans *Proceedings of the 40th International Conference on Parallel Processing (ICPP-2011)*, Taipei, Taiwan, septembre 2011.

[118] COHEN (D.), TALPEY (T.), KANEVSKY (A.) *et al.*, « Remote Direct Memory Access over the Converged Enhanced Ethernet Fabric : Evaluating the Options », dans *Proceedings of the 17th Annual Symposium on High-Performance Interconnects (HotI'09)*, p. 123–130, New York, NJ, août 2009.

[119] RABENSEIFNER (R.), HAGER (G.) et JOST (G.), « Hybrid MPI/OpenMP Parallel Programming on Clusters of Multi-Core SMP Nodes », dans *Proceedings of the 17th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP 2009)*, p. 427–436, Weimar, Allemagne, février 2009.

[120] CAPPELLO (F.) et ETIEMBLE (D.), « MPI versus MPI+OpenMP on the IBM SP for the NAS Benchmarks », dans *Proceedings of the 14th International Conference on Supercomputing (ICS'00)*, Dallas, TX, novembre 2000.

[121] RABENSEIFNER (R.), « Hybrid Parallel Programming : Performance Problems and Chances », dans *Proceedings of the 45th Cray User Group (CUG) Conference*, Columbus, Ohio, mai 2003.

[122] RABENSEIFNER (R.), « Hybrid Parallel Programming on HPC Platforms », dans *Proceedings of the Fifth European Workshop on OpenMP (EWOMP)*, Aachen, Allemagne, septembre 2003.

[123] BASUMALLIK (A.), MIN (S.-J.) et EIGENMANN (R.), « Programming Distributed Memory Systems Using OpenMP », dans *Proceedings of the International Workshop on High-Level Parallel Programming Models and Supportive Environments (HIPS-TOPMoDRS), tenu conjointement avec IPDPS'07*, p. 181, Long Beach, CA, mars 2007.

[124] HESS (M.), JOST (G.), MÜLLER (M.) et RÜHLE (R.), « Experiences using OpenMP based on compiler directed software DSM on a PC cluster », dans *Proceedings of the OpenMP applications and tools 2003 international conference on OpenMP shared memory parallel programming*, coll. « WOMPAT'03 », p. 211–226, Toronto, Canada, 2003. Springer-Verlag.

[125] JOST (G.), JIN (H.), MEY (D. A.) et HATAY (F. F.), « Comparing the OpenMP, MPI, and Hybrid Programming Paradigms on an SMP Cluster », dans *Proceedings of the Fifth European Workshop on OpenMP (EWOMP)*, Aachen, Allemagne, septembre 2003.

[126] KRAWEZIK (G.) et CAPPELLO (F.), « Performance Comparison of MPI and three OpenMP Programming Styles on Shared Memory Multiprocessors », dans *Proceedings of the fifteenth annual ACM Symposium on Parallel Algorithms and Architectures (SPAA2003)*, p. 118–127, San Diego, CA, juin 2003.

[127] BALAJI (P.), BUNTINAS (D.), GOODELL (D.) *et al.*, « Toward Efficient Support for Multithreaded MPI Communication », dans *Proceedings of the 15th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface (EuroPVM/MPI 2008)*, p. 120–129, Dublin, Irelande, septembre 2008. Springer-Verlag.

[128] INTEL. « Thread Affinity Interface (Linux* and Windows*) ».

[129] INTEL. « Thread Building Blocks ». `http://www.intel.com/software/products/tbb/`.

[130] FRIGO (M.), LEISERSON (C. E.) et RANDALL (K. H.), « The Implementation of the Cilk-5 Multithreaded Language », dans *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, Montréal, Canada, juin 1998.

[131] MERCIER (G.) et CLET-ORTEGA (J.), « Towards an Efficient Process Placement Policy for MPI Applications in Multicore Environments », dans *Proceedings of the 16th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface (EuroPVM/MPI 2009)*, vol. 5759 (coll. *Lecture Notes in Computer Science*), p. 104–115, Espoo, Finlande, septembre 2009. Springer.

[132] JEANNOT (E.) et MERCIER (G.), « Near-optimal placement of MPI processes on hierarchical NUMA architecture », dans *Proceedings of the 16th European Conference on Parallel and Distributed Computing (Euro-Par'10)*, Ischia, Italie, août 2010.

[133] CHEN (H.), CHEN (W.), HUANG (J.) *et al.*, « MPIPP : an automatic profile-guided parallel process placement toolset for SMP clusters and multiclusters », dans *Proceedings of the 20th International Conference on Supercomputing (ICS'06)*, p. 353–360, 2006.

[134] DAVID SOLT. « A profile based approach for topology aware MPI rank placement », 2007. `http://www.tlc2.uh.edu/hpcc07/Schedule/speakers/hpcc_hp-mpi_solt.ppt`.

[135] DUESTERWALD (E.), WISNIEWSKI (R. W.), SWEENEY (P. F.) *et al.* « Method and System for Optimizing Communication in MPI Programs for an Execution Environment ». Brevet, 2008. `http://www.faqs.org/patents/app/20080288957`.

[136] ZHANG (J.), ZHAI (J.), CHEN (W.) et ZHENG (W.), « Process Mapping for MPI Collective Communications », dans *Proceedings of the 15th European Conference on Parallel and Distributed Computing (Euro-Par'09)*, p. 81–92, Delft, Pays Bas, 2009. Springer-Verlag.

[137] SANYAL (S.), JAIN (A.), DAS (S.) et BISWAS (R.), « A hierarchical and distributed approach for mapping large applications to heterogeneous grids using genetic algorithms », dans *Proceedings of the International Conference on Cluster Computing (Cluster'03)*, p. 496 – 499. IEEE Computer Society, décembre 2003.

[138] PHINJAROENPHAN (P.), BEVINAKOPPA (S.) et ZEEPHONGSEKUL (P.), « A Heuristic Algorithm for Mapping Parallel Applications on Computational Grids », dans SLOOT (P. M. A.), HOEKSTRA (A. G.), PRIOL (T.) *et al.*, éditeurs, *Advances in Grid Computing - EGC 205*, vol. 3470 (coll. *Lecture Notes in Computer Science*), p. 3–5. Springer Berlin / Heidelberg, 2005.

[139] BHANOT (G.), GARA (A.), HEIDELBERGER (P.) *et al.*, « Optimizing task layout on the Blue Gene/L supercomputer », *IBM Journal of Research and Development*, vol. 49, n° 2.3, mars 2005, p. 489 –500.

[140] YU (H.), CHUNG (I.-H.) et MOREIRA (J.), « Topology mapping for Blue Gene/L supercomputer », dans *Proceedings of the 20th International Conference on Supercomputing (ICS'06)*, Tampa, FL, 2006. ACM.

[141] PELLEGRINI (F.). « Scotch and libScotch 5.1 User's Guide ». Manuel d'utilisation, août 2008.

[142] BUNTINAS (D.), MERCIER (G.) et GROPP (W.), « Data Transfers between Processes in an SMP System : Performance Study and Application to MPI », *Conference on Parallel Processing (ICPP)*, août 2006, p. 487–496.

[143] TRAÄF (J. L.), « Implementing the MPI Process Topology Mechanism », dans *Proceedings of the 16th International Conference on Supercomputing (ICS'02)*, p. 28, Los Alamitos, CA, USA, 2002. IEEE Computer Society.

[144] KRANZ (D.), JOHNSON (K.), AGARWAL (A.) *et al.*, « Integrating message-passing and shared-memory : Early experience », dans *Proceedings of the 4th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, p. 54–63, San Diego, CA, 1993.

[145] BRIGHTWELL (R.), HUDSON (T.) et PEDRETTI (K.), « SMARTMAP : Operating System Support for Efficient Data Sharing Among Processes on a Multi-Core Processor », dans *Proceedings of the ACM/IEEE Conference on High Performance Networking and Computing (SC 2008)*, Austin, TX, novembre 2008. ACM Press.

[146] INTEL. « Accelerating High-Speed Networking with Intel I/O Acceleration Technology ». White paper, mai 2005. `http://download.intel.com/support/network/sb/98856.pdf`.

[147] GROVER (A.) et LEECH (C.), « Accelerating Network Receive Processing (Intel I/O Acceleration Technology) », dans *Proceedings of the Linux Symposium (OLS2005)*, p. 281–288, Ottawa, Canada, juillet 2005.

[148] VAIDYANATHAN (K.) et PANDA (D. K.), « Benefits of I/O Acceleration Technology (I/OAT) in Clusters », dans *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS-2007)*, p. 220–229, San Jose, CA, avril 2007.

[149] VAIDYANATHAN (K.), HUANG (W.), CHAI (L.) et PANDA (D. K.), « Designing Efficient Asynchronous Memory Operations Using Hardware Copy Engine : A Case Study with I/OAT », dans *Proceedings of the 7th Workshop on Communication Architecture for Clusters (CAC'07), tenu conjointement avec IPDPS'07*, p. 234, Long Beach, CA, mars 2007.

[150] VAIDYANATHAN (K.), CHAI (L.), HUANG (W.) et PANDA (D. K.), « Efficient Asynchronous Memory Copy Operations on Multi-Core Systems and I/OAT », dans *Proceedings of the International Conference on Cluster Computing (Cluster'07)*, p. 159–168, Austin, TX, septembre 2007. IEEE Computer Society.

[151] KIELMANN (T.), HOFMAN (R. F. H.), BAL (H. E.) *et al.*, « MagPIe : MPI's collective communication operations for clustered wide area systems », dans *Proceedings of the ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, mai 1999.

[152] KIELMANN (T.), BAL (H.) et GORLATCH (S.), « Bandwidth-efficient collective communication for clustered wide area systems », dans *Proceedings of the 14th International Parallel and Distributed Processing Symposium, (IPDPS'00)*, p. 492 –499. IEEE Computer Society, 2000.

[153] KIELMANN (T.), HOFMAN (R. F. H.), BAL (H. E.) *et al.*, « MPI's reduction operations in clustered wide area systems », dans *Proceedings of the Message Passing Interface Developer's and User's Conference (MPIDC'99)*, p. 43–52, 1999.

[154] KARONIS (N.), SUPINSKI (B. R. D.), FOSTER (I.) *et al.*, « Exploiting Hierarchy in Parallel Computer Networks to Optimize Collective Operation Performance », dans *Proceedings of the 14th International Symposium on Parallel and Distributed Processing*, p. 377–384, Washington, DC, USA, 2000. IEEE Computer Society.

[155] HUSBANDS (P.) et HOE (J.), « MPI-StarT : Delivering Network Performance to Numerical Applications », dans *Proceedings of the 12th International Conference on Supercomputing (ISC'98)*, p. 17, novembre 1998.

[156] MATSUDA (M.), KUDOH (T.), KODAMA (Y.) *et al.*, « Efficient MPI Collective Operations for Clusters in Long-and-Fast Networks », dans *Proceedings of the International Conference on Cluster Computing (Cluster'06)*. IEEE Computer Society, septembre 2006.

[157] VADHIYAR (S. S.), FAGG (G. E.) et DONGARRA (J.), « Automatically tuned collective communications », dans *Proceedings of the 14th International Conference on Supercomputing (ICS'00)*, Dallas, TX, USA, 2000. IEEE Computer Society.

[158] BARNETT (M.), PAYNE (D. G.) et GEIJN (R. V. D.), « Optimal Broadcasting in Mesh-Connected Architectures ». Rapport technique, Austin, TX, USA, 1991.

[159] BARNETT (M.), LITTLEFIELD (R.), PAYNE (D.) et VAN DE GEIJN (R.), « Global combine on mesh architectures with wormhole routing », dans *Proceedings of the 7th International Parallel Processing Symposium (IPDPS'93)*, p. 156 –162. IEEE Computer Society, avril 1993.

[160] BARNETT (M.), PAYNE (D. G.), VAN DE GEIJN (R. A.) et WATTS (J.), « Broadcasting on meshes with wormhole routing », *Journal of Parallel and Distributed Computing*, vol. 35, juin 1996, p. 111–122.

[161] BOKHARI (S.) et BERRYMAN (H.), « Complete exchange on a circuit switched mesh », dans *Proceedings of the Scalable High Performance Computing Conference (SHPCC-92)*, p. 300 –306, avril 1992.

[162] SCOTT (D.), « Efficient All-to-All Communication Patterns in Hypercube and Mesh Topologies », dans *Proceedings of the 6th Distributed Memory Computing Conference*, p. 398 –403, avril 1991.

[163] RABENSEIFNER (R.), « Optimization of Collective Reduction Operations », dans *International Conference on Computational Science*, p. 1–9, 2004.

[164] TRÄFF (J.), « Hierarchical gather/scatter algorithms with graceful degradation », dans *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04)*, p. 80. IEEE Computer Society, avril 2004.

[165] TRÄFF (J.), « Efficient Allgather for Regular SMP-Clusters », dans MOHR (B.), TRÄFF (J.), WORRINGEN (J.) et DONGARRA (J.), éditeurs, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, vol. 4192 (coll. *Lecture Notes in Computer Science*), p. 58–65. Springer Berlin / Heidelberg, 2006.

[166] SANDERS (P.) et TRÄFF (J.), « The hierarchical factor algorithm for all-to-all communication », dans MONIEN (B.) et FELDMANN (R.), éditeurs, *Proceedings of the 8th European Conference on Parallel and Distributed Computing (Euro-Par'02)*, vol. 2400 (coll. *Lecture Notes in Computer Science*), p. 17–51. Springer Berlin / Heidelberg, 2002.

[167] BRUCK (J.), HO (C.-T.), KIPNIS (S.) et WEATHERSBY (D.), « Efficient algorithms for all-to-all communications in multi-port message-passing systems », dans *Proceedings of the 6th annual ACM symposium on Parallel Algorithms and Architectures*, coll. « SPAA '94 », p. 298–309, Cape May, NJ, USA, 1994. ACM.

[168] JOHNSSON (S. L.) et HO (C.-T.), « Interconnection networks for high-performance parallel computers », chap. Optimum broadcasting and personalized communication in hypercubes, p. 363–382. IEEE Computer Society, Los Alamitos, CA, USA, 1994.

[169] BARNETT (M.), SHULER (L.), VAN DE GEIJN (R.) *et al.*, « Interprocessor collective communication library (InterCom) », dans *Proceedings of the Scalable High-Performance Computing Conference*, p. 357 –364, mai 1994.

[170] MITRA (P.), PAYNE (D.), SHULER (L.) *et al.*, « Fast Collective Communication Libraries, Please ». Rapport technique, Austin, TX, USA, 1995.

[171] YANG (Y.) et WANG (J.), « Pipelined all-to-all broadcast in all-port meshes and tori », *IEEE Transactions on Computers*, vol. 50, n° 10, octobre 2001, p. 1020 –1032.

[172] CHAN (E.), HEIMLICH (M.), PURKAYASTHA (A.) et VAN DE GEIJN (R.), « On optimizing collective communication », dans *Proceedings of the International Conference on Cluster Computing (Cluster'04)*, p. 145 – 155. IEEE Computer Society, septembre 2004.

[173] BALA (V.), BRUCK (J.), CYPHER (R.) *et al.*, « CCL : A Portable and Tunable Collective Communication Library for Scalable Parallel Computers », *IEEE Transactions on Parallel and Distributed Systems*, vol. 6, 1995, p. 154–164.

[174] FAGG (G.), VADHIYAR (S.) et DONGARRA (J.), « ACCT : Automatic Collective Communications Tuning », dans DONGARRA (J.), KACSUK (P.) et PODHORSZKI (N.), éditeurs, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, vol. 1908 (coll. *Lecture Notes in Computer Science*), p. 354–361. Springer Berlin / Heidelberg, 2000.

[175] PJEŠIVAC-GRBOVIĆ (J.), BOSILCA (G.), FAGG (G. E.) *et al.*, « MPI collective algorithm selection and quadtree encoding », *Parallel Computing*, vol. 33, n° 9, 2007, p. 613 – 623. Selected Papers from EuroPVM/MPI 2006.

[176] THAKUR (R.), « Improving the performance of collective operations in MPICH », dans *Proceedings of the 10th European PVM/MPI User's Group Meeting on Recent Advances*

in Parallel Virtual Machine and Message Passing Interface (EuroPVM/MPI 2003), coll. « Lecture Notes in Computer Science », p. 257–267. Springer Verlag, 2003.

[177] THAKUR (R.) et RABENSEIFNER (R.), « Optimization of Collective Communication Operations in MPICH », *International Journal of High Performance Computing Applications*, vol. 19, 2005, p. 49–66.

[178] MAMIDALA (A. R.), KUMAR (R.), DE (D.) et PANDA (D. K.), « MPI Collectives on Modern Multicore Clusters : Performance Optimizations and Communication Characteristics », dans *Proceedings of the Int'l Symposium on Cluster Computing and the Grid (CCGrid)*, Lyon, France, mai 2008. IEEE Computer Society.

[179] KANDALLA (K.), SUBRAMONI (H.), SANTHANARAMAN (G.) *et al.*, « Designing Multi-Leader-Based Allgather Algorithms for Multi-Core Clusters », dans *Proceedings of the 9th Workshop on Communication Architecture for Clusters (CAC'09), tenu conjointement avec IPDPS'09*, Rome, Italie, mai 2009. IEEE Computer Society.

[180] GRAHAM (R. L.) et SHIPMAN (G.), « MPI Support for Multi-core Architectures : Optimized Shared Memory Collectives », dans *Proceedings of the 15th European PVM/MPI Users' Group Meeting on Recent Advances in Parallel Virtual Machine and Message Passing Interface (EuroPVM/MPI 2008)*, p. 130–140, Dublin, Irelande, 2008. Springer-Verlag.

[181] KANDALLA (K.), SUBRAMONI (H.), VISHNU (A.) et PANDA (D. K.), « Designing topology-aware collective communication algorithms for large scale InfiniBand clusters : Case studies with Scatter and Gather », dans *Proceedings of the 10th Workshop on Communication Architecture for Clusters (CAC'10), tenu conjointement avec IPDPS'10*, Atlanta, GA, avril 2010. IEEE Computer Society.

[182] KUMAR (R.), MAMIDALA (A.) et PANDA (D. K.), « Scaling Alltoall Collective on Multi-core Systems », dans *Workshop on Communication Architecture for Clusters, tenu conjointement avec IPDPS'08*, Miami, FL, avril 2008. IEEE Computer Society.

[183] TIPPARAJU (V.), NIEPLOCHA (J.) et PANDA (D.), « Fast Collective Operations Using Shared and Remote Memory Access Protocols on Clusters », dans *Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS'03)*. IEEE Computer Society, avril 2003.

[184] GUPTA (R.), TIPPARAJU (V.), NIEPLOCHA (J.) et PANDA (D.), « Efficient barrier using remote memory operations on VIA-based clusters », dans *Proceedings of the International Conference on Cluster Computing (Cluster'02)*, p. 83 – 90. IEEE Computer Society, 2002.

[185] WU (M.-S.), KENDALL (R. A.) et ALURU (S.), « Exploring collective communications on a cluster of smps », dans *HPCASIA '04 : Proceedings of the High Performance Computing and Grid in Asia Pacific Region, Seventh International Conference*, p. 114–117, Washington, DC, USA, 2004. IEEE Computer Society.

[186] WU (M.-S.), KENDALL (R.) et WRIGHT (K.), « Optimizing collective communications on SMP clusters », dans *Proceedings of the International Conference on Parallel Processing (ICPP-2005)*, p. 399 – 407, juin 2005.

[187] WU (M.-S.), KENDALL (R. A.) et ALURU (S.), « A tunable collective communication framework on a cluster of smps », dans *Proceedings of the IASTED International Conference on Parallel and Distributed Computing and Networks*, p. 56–63, 2004.

[188] SISTARE (S.), VANDEVAART (R.) et LOH (E.), « Optimization of mpi collectives on clusters of large-scale smp's », dans *Proceedings of the 13th International Conference on Supercomputing (ICS'99)*, p. 23, Portland, OR, USA, 1999. ACM.

[189] MAMIDALA (A. R.), CHAI (L.), WOOK JIN (H.) *et al.*, « Efficient SMP-aware MPI-level broadcast over InfiniBand's hardware multicast », dans *Proceedings of the 6th Workshop on Communication Architecture for Clusters (CAC'06), tenu conjointement avec IPDPS'06*, Rhodes, Grèce, 2006.

[190] LIU (J.), MAMIDALA (A.) et PANDA (D.), « Fast and scalable MPI-level broadcast using InfiniBand's hardware multicast support », dans *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04)*, p. 10. IEEE Computer Society, avril 2004.

[191] MAMIDALA (A.), LIU (J.) et PANDA (D.), « Efficient Barrier and Allreduce on Infiniband clusters using multicast and adaptive algorithms », dans *Proceedings of the International Conference on Cluster Computing (Cluster'04)*, p. 135 – 144. IEEE Computer Society, septembre 2004.

[192] MAMIDALA (A. R.), JIN (H.-W.) et PANDA (D. K.), « Efficient Hardware Multicast Group Management for Multiple MPI Communicators over InfiniBand », dans MARTINO (B. D.), KRANZLMÜLLER (D.) et DONGARRA (J.), éditeurs, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, vol. 3666 (coll. *Lecture Notes in Computer Science*), p. 388–398. Springer Berlin / Heidelberg, 2005.

[193] KINI (S.), LIU (J.), WU (J.) *et al.*, « Fast and Scalable Barrier Using RDMA and Multicast Mechanisms for InfiniBand-Based Clusters », dans *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, vol. 2840 (coll. *Lecture Notes in Computer Science*), p. 369–378. Springer Berlin / Heidelberg, 2003.

[194] MAMIDALA (A. R.), VISHNU (A.) et PANDA (D. K.), « Efficient Shared Memory and RDMA Based Design for MPI_Allgather over InfiniBand », dans MOHR (B.), TRÄFF (J.), WORRINGEN (J.) et DONGARRA (J.), éditeurs, *Recent Advances in Parallel Virtual Machine and Message Passing Interface*, vol. 4192 (coll. *Lecture Notes in Computer Science*), p. 66–75. Springer Berlin / Heidelberg, 2006.

[195] TU (B.), ZOU (M.), ZHAN (J.) *et al.*, « Multi-core aware optimization for MPI collectives », dans *Proceedings of the International Conference on Cluster Computing (Cluster'08)*, p. 322 –325. IEEE Computer Society, octobre 2008.

[196] COTI (C.), HERAULT (T.) et CAPPELLO (F.), « MPI Applications on Grids : A Topology Aware Approach », dans SIPS (H.), EPEMA (D.) et LIN (H.-X.), éditeurs, *Proceedings of the 16th European Conference on Parallel and Distributed Computing (Euro-Par'09)*, vol. 5704 (coll. *Lecture Notes in Computer Science*), p. 466–477. Springer Berlin / Heidelberg, 2009.

[197] FARAJ (A.), KUMAR (S.), SMITH (B.) *et al.*, « MPI Collective Communications on The Blue Gene/P Supercomputer : Algorithms and Optimizations », dans *Proceedings of the 17th Symposium on High Performance Interconnects (HOTI'09)*, p. 63 –72. IEEE Computer Society, août 2009.

[198] TUDUCE (I.), MAJO (Z.), GAUCH (A.) *et al.*, « Asymetries in Multi-Core Systems – Or Why We Need Better Performance Measurement Units », dans *Proceedings of the Exascale*

Evaluation and Research Techniques Workshop (EXERT 2010), tenu conjointement avec ASPLOS 2010, Pittsburg, PA, mars 2010. ACM.

[199] KADAYIF (I.) et KANDEMIR (M.), « Data space-oriented tiling for enhancing locality », *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 4, mai 2005, p. 388–414.

[200] WERSTEIN (P.), PETHICK (M.) et HUANG (Z.), « A performance comparison of DSM, PVM, and MPI », dans *Proceedings of the 4h International Conference on Parallel and Distributed Computing, Applications and Technologies (PDCAT'2003)*, p. 476 – 482, août 2003.

[201] GEIST (A.), BEGUELIN (A.), DONGARRA (J.) *et al.*, *Parallel Virtual Machine : A Users' Guide and Tutorial for Networked Parallel Computing*. MIT Press, 1994.

[202] LIU (X.), « Performance Evaluation of a Hardware Implementation of VIA ». Rapport technique, Department of Computer Science and Engineering University of California, San Diego, juin 1999.

[203] SPEIGHT (E.), ABDEL-SHAFI (H.) et BENNETT (J. K.), « Realizing the Performance Potential of the Virtual Interface Architecture », dans *Proceedings of the 13th International Conference on Supercomputing (ICS'99)*, p. 184–192, 1999.

[204] BANIKAZEMI (M.), ABALI (B.) et PANDA (D. K.), « Comparison and Evaluation of Design Choices for Implementing the Virtual Interface Architecture (VIA) », dans *Network-Based Parallel Computing. Communication, Architecture, and Applications*, vol. 1797 (coll. *Lecture Notes in Computer Science*), p. 145–161. Springer Berlin / Heidelberg, 2000.

[205] BRIGHTWELL (R.) et MACCABE (A.), « Scalability limitations of VIA-based technologies in supporting MPI », dans *Proceedings of the 4th MPI Developer's and User's Conference*, mars 2000.

[206] NIEPLOCHA (J.) et CARPENTER (B.), « ARMCI : A portable remote memory copy library for distributed array libraries and compiler run-time systems », dans *Parallel and Distributed Processing*, vol. 1586 (coll. *Lecture Notes in Computer Science*), p. 533–546. Springer Berlin / Heidelberg, 1999.

[207] CARLSON (W.), DRAPER (J.), CULLER (D.) *et al.*, « Introduction to upc and language specification ». Rapport technique n° CCS-TR-99-157, George Mason University, mai 1999.

[208] HOEFLINGER (J.), « Extending OpenMP to clusters », *White Paper, Intel Corporation*, 2006.

Liste des publications

Conférences internationales avec publication des actes et comité de lecture

[1] BROQUEDIS (F.), CLET-ORTEGA (J.), MOREAUD (S.) *et al.*, « hwloc : a Generic Framework for Managing Hardware Affinities in HPC Applications », dans *Proceedings of the 18th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP2010)*, p. 180–186, Pise, Italie, février 2010. IEEE Computer Society Press.

[2] BUNTINAS (D.), GOGLIN (B.), GOODELL (D.) *et al.*, « Cache-Efficient, Intranode Large-Message MPI Communication with MPICH2-Nemesis », dans *Proceedings of the 38th International Conference on Parallel Processing (ICPP-2009)*, p. 462–469, Vienne, Autriche, septembre 2009. IEEE Computer Society Press.

[3] MOREAUD (S.) et GOGLIN (B.), « Impact of NUMA Effects on High-Speed Networking with Multi-Opteron Machines », dans *The 19th IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2007)*, Cambridge, MA, novembre 2007.

Colloques internationaux avec publication des actes et comité de lecture

[4] MOREAUD (S.), GOGLIN (B.), GOODELL (D.) et NAMYST (R.), « Optimizing MPI Communication within large Multicore nodes with Kernel assistance », dans *CAC 2010 : The 10th Workshop on Communication Architecture for Clusters, held in conjunction with IPDPS 2010*, Atlanta, GA, avril 2010. IEEE Computer Society Press.

[5] MOREAUD (S.), GOGLIN (B.) et NAMYST (R.), « Adaptive MPI Multirail Tuning for Non-Uniform Input/Output Access », dans *Proceedings of the 17th European MPI Users Group Conference*, coll. « Lecture Notes in Computer Science », Stuttgart, Allemagne, septembre 2010. Springer.

[6] GOGLIN (B.) et MOREAUD (S.), « Dodging Non-Uniform I/O Access in Hierarchical Collective Operations for Multicore Clusters », dans *CASS 2011 : The 1st Workshop on Communication Architecture for Scalable Systems, held in conjunction with IPDPS 2011*, Anchorage, AK, mai 2011. IEEE Computer Society Press.

Conférences nationales avec publication des actes et comité de lecture

[7] MOREAUD (S.), « Adaptation des communications MPI intra-nœud aux architectures multicœurs modernes », dans *19ème Rencontres Francophones du Parallélisme*, Toulouse, France, septembre 2009.

[8] MOREAUD (S.), « Impacts des effets NUMA sur les communications haute performance dans les grappes de calcul », dans *18ème Rencontres Francophones du Parallélisme*, Fribourg, Suisse, février 2008. École d'ingénieurs et d'architectes de Fribourg.

Rapports de recherche

[9] MOREAUD (S.), « Impact des architectures multiprocesseurs sur les communications dans les grappes de calcul : de l'exploration des effets numa au placement automatique ». Mémoire de Master Recherche, Université Bordeaux 1, juin 2007.