

Adaptation des communications MPI intra-nœud aux architectures multicœurs modernes

Stéphanie Moreaud
Laboratoire Bordelais de Recherche en Informatique - INRIA
Stephanie.Moreaud@labri.fr

Résumé

L'émergence des processeurs multicœurs accroît les besoins en transferts de données entre les processus à l'intérieur des machines. Comme la plupart des implémentations portables de MPI, MPICH2 utilise un schéma de communication intra-nœud reposant sur plusieurs recopies mémoire. Ce modèle souffre d'une utilisation intensive des processeurs et d'une forte pollution de cache limitant significativement les performances. Grâce à l'interface de programmation *Large Message Transfer* de MPICH2, conçue pour supporter un vaste panel de mécanismes de transfert, il est cependant possible de modifier cette stratégie. La mise en place d'une stratégie de copie directe basée sur l'appel système `vmsplice` de Linux permet d'améliorer les performances dans certains cas. Nous présentons une seconde stratégie de copie directe, reposant sur un module noyau dédié nommé KNEM. Il tire profit des capacités matérielles de déport de copie mémoire, en les activant dynamiquement selon les caractéristiques physiques des caches et de la taille des messages. Cette nouvelle solution surpasse les méthodes de transfert habituelles et la stratégie `vmsplice`, lorsque les cœurs sur lesquels s'exécutent les processeurs ne partagent aucun cache, ou pour des transferts de très larges messages. Les opérations de communication collectives montrent quant à elles une amélioration spectaculaire, et le test NAS IS obtient une accélération de 25% et une meilleure utilisation des caches.

Mots-clés : Communications MPI intra-nœud, multicœurs, partage de cache, placement, MPICH2.

1. Introduction

Les architectures multicœurs sont omniprésentes dans le paysage du calcul parallèle, avec un nombre de cœurs en augmentation et une organisation de plus en plus complexe des ressources mémoire. En l'absence de consensus sur ce que le modèle de programmation le plus adapté pour les architectures massivement multicœurs devrait être, MPI continue à jouer un rôle prépondérant dans le développement des applications parallèles. Améliorer les performances des implémentations MPI pour les communications intra-nœud en mémoire partagée est donc plus que jamais d'actualité et a déjà fait l'objet de nombreuses recherches [5, 9].

Les bibliothèques de communication doivent être capables de tirer pleinement avantage des nouvelles caractéristiques des architectures multicœurs complexes. Par exemple, les cœurs localisés sur un même nœud vont partager non seulement de la mémoire, mais aussi une partie de la hiérarchie de cache. Les implémentations MPI vont de plus généralement utiliser des mécanismes de communication différents pour de petits messages nécessitant une faible latence, et pour de larges messages pour lesquels une bande passante importante est primordiale. L'optimisation des communications par petits messages a déjà été étudiée [4] et nous nous focalisons dans cet article sur les communications intra-nœud par gros messages (au moins de l'ordre de la dizaine de kilo-octets). Plusieurs méthodes pour ce type de communication en mémoire partagée ont été évaluées [3], incluant notamment les copies mémoire par *double buffering* ou via un module noyau.

Après avoir présenté les stratégies de communications par larges messages dans MPICH2 en Section 2, nous décrivons un nouveau mécanisme de copie directe basé sur un module noyau dédié nommé KNEM en Section 3. Nous évaluons ensuite les performances de ce nouveau mécanisme en tenant compte de la bande passante et de la pollution de cache en Section 4 avant de conclure.

2. Communications MPI intra-nœud

Nous décrivons dans cette section les méthodes de communication intra-nœud par mémoire partagée entre processus MPI, en commençant par la stratégie *double-buffering* communément utilisée. Nous passons ensuite aux stratégies de copie directe mises en place dans MPICH2 grâce à son interface de transfert de larges messages.

2.1. Le *double-buffering*

Le transfert de messages entre processus MPI d'un même nœud repose communément sur le modèle de la mémoire partagée. Deux processus communicant allouent un tampon partagé, le processus émetteur copie ensuite ses données dans le tampon, puis le processus récepteur les copie à son tour depuis le tampon vers sa zone mémoire destination. Ce procédé utilisé dans MPICH2 [5], OPEN MPI [7] et MVAPICH [12] requiert ainsi deux copies et nécessite une synchronisation pour assurer que la lecture du tampon intermédiaire 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 d'une part avoir un impact négatif sur la mémoire disponible, et d'autre part impliquer une forte latence inhérente au fait que le processus récepteur doit attendre que la totalité des données ait été copiée dans le tampon partagé avant de commencer la copie depuis celui-ci. Pour limiter ces inconvénients, on 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. Cette approche nommée *double-buffering* réduit la latence, le processus récepteur n'ayant pas à attendre la fin de l'écriture du message entier avant de commencer sa lecture. Elle améliore la bande passante grâce à l'exécution des deux copies en parallèle par les deux processus mis en jeu. Les performances globales sont fonction de la taille des tampons : 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 réduite. MPICH2 utilise des tampons de 64 ko, bon compromis entre latence et débit.

2.2. Transferts de larges messages avec MPICH2-Nemesis

MPICH2 est une implémentation portable haute performance de la version 2.1 du standard *Message Passing Interface* (MPI) [10]. La version 1.1 de MPICH2 utilise un sous-système de communication appelé NEMESIS [5], qui effectue les communications intra-nœud par mémoire partagée et inter-nœuds via le réseau. La communication en mémoire partagée de NEMESIS a été optimisée pour minimiser la latence des petits messages et maximiser la bande passante des grands messages. L'une des innovations pour le transfert de grands messages est l'introduction de l'interface de programmation *Large Message Transfer* (LMT), conçue suffisamment généralement pour supporter un vaste panel de mécanismes de communication. L'interface LMT supporte en particulier les méthodes par accès mémoire à distance (RDMA par exemple) ou par communication explicite (au travers d'un tube UNIX, ou *double buffering*). Cette flexibilité permet à NEMESIS d'utiliser le meilleur mécanisme disponible sur chaque plateforme et pour chacun des transferts.

La méthode du *double buffering* utilisée initialement par Nemesis, a le désavantage d'utiliser deux processeurs, les rendant de ce fait inaccessibles pour des phases de calcul. Elle entraîne également une forte pollution des caches en y remplaçant des données utiles à l'application par les données en cours de transfert. Une solution évidemment préférable serait 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. Pour pouvoir mettre en place une copie directe, il faut donc solliciter le système d'exploitation. Une première méthode, basée sur l'appel système `vmsplICE` [1], introduit dans le noyau LINUX 2.6.17, a été implémentée dans NEMESIS. Les processus émetteur et récepteur ouvrent un même tube UNIX dans lequel le processus émetteur va *attacher* des pages mémoire grâce à l'appel `vmsplICE` (plutôt que de les copier par `writEV`). Lorsque le processus récepteur utilise l'appel système `readV`, les données sont alors directement copiées de ces pages vers la zone mémoire destination. Cette solution apporte dans certaines circonstances un gain de performance par rapport au *double-buffering* qui est cependant plus portable ¹.

¹ Disponible sur l'ensemble des systèmes dotés de l'appel système `mmap` ou de mémoire partagée System V.

Ces méthodes ne sont pas toujours toutes disponibles et présentent des performances plus ou moins élevées sur les différentes plateformes selon le type de communication mis en jeu, la taille des messages et la répartition des processus sur les cœurs de la machine. L'utilisation de l'interface LMT permet à NEMESIS de choisir dynamiquement le mécanisme disponible le plus adapté à chaque situation.

3. Conception d'un mécanisme de copie directe optimisée

Cette section détaille la conception d'un système de copie directe basée sur le module noyau dédié KNEM qui permet notamment le déport de copie mémoire dans le matériel. On discute ensuite les critères utilisés pour choisir la meilleure stratégie selon le message et les caractéristiques matérielles.

3.1. KNEM : un LMT dédié dans le noyau LINUX

Si le LMT `vmsplice` à l'avantage de fonctionner sur tous les systèmes LINUX récents, il supporte cependant uniquement une interface synchrone et bloquante côté récepteur. Ces restrictions nous ont amenés développer une stratégie de copie directe basée sur un module noyau dédié. L'objectif est d'offrir de meilleures performances que `vmsplice` grâce à un modèle adapté aux communications MPI.

Cette idée a été précédemment étudiée dans des implémentations spécifiques au matériel d'interconnexion. MX [11] et BIP-SMP [13] s'appuient par exemple sur les réseaux Myrinet, tandis que OpenMX [6] requière un réseau Ethernet pour les communications inter-noeud et permet l'utilisation déport de copie mémoire dans le matériel. LIMIC2 [9] offre une implémentation multi-support mais nous proposons d'aller plus loin en supportant notamment les communications non-bloquantes, asynchrones, vectorielles, et en profitant des fonctionnalités modernes de déport de copie mémoire dans le matériel.

Le module LINUX KNEM (*Kernel Nemesis*) s'appuie sur un périphérique caractère (`/dev/knem`) qui implémente deux principales commandes de communication (Figure 1). 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 à travers 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.

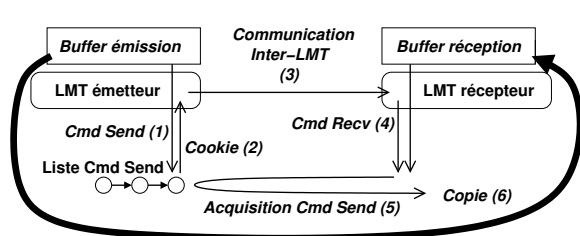


FIG. 1 – Transfert de grands messages avec le module noyau KNEM.

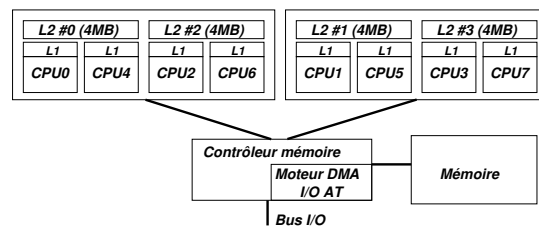


FIG. 2 – Architecture à deux processeurs quadricœurs XEON dotée de la technologie I/OAT et de caches L2 de 4 Mo partagés entre paires de cœurs.

3.2. Déport de copie mémoire dans KNEM

La technologie *Input/Output Acceleration Technology* (I/OAT) offre un ensemble de fonctions implémentées dans les contrôleurs mémoire INTEL récents [8]. Elle contient entre autre 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 travail au profit de travail "utile". Ce modèle empêche par la même occasion la pollution du cache, exonéré des données relatives à la copie. Pour profiter de ces avantages, nous avons ajouté au module KNEM la possibilité d'utiliser le déport de copie I/OAT. Lorsque le moteur DMA est disponible, cette option peut être activée par le LMT KNEM via un paramètre ajouté à la commande de réception.

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 (utilisée par le matériel et donc par I/OAT). En effet, 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 produit pendant une copie I/OAT, les données manipulées deviendraient invalides. Pour assurer que la projection mémoire ne changera 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 également appliquée en l'absence d'I/OAT car 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 des problèmes de verrouillage sous-jacents.

3.3. Choix de la stratégie à utiliser

KNEM est supposé offrir de meilleures performances que `vmsplice` puisqu'il a été entièrement conçu pour le transfert de message MPI alors que `vmsplice` est une solution plus générale. Lorsque le chargement d'un module noyau est une contrainte acceptable, un utilisateur devrait donc utiliser préférentiellement KNEM. Si ce n'est pas le cas, l'utilisation du LMT `vmsplice` sur les systèmes LINUX pourrait être préférable au *double-buffering* car plus efficace grâce à la réduction du nombre de copies.

L'utilisation de KNEM pose la question de savoir quand déléguer la copie mémoire à I/OAT en tâche de fond plutôt que d'utiliser la copie régulière par le processeur. En examinant les performances sur des processeurs XEON 2,33 GHz dotés de caches L2 partagés de 4 Mo entre chaque paire de cœurs (Figure 2), nous avons observé que le seuil de transition idéal semble proche de 1 Mo. En effectuant les mêmes tests sur des cœurs ne partageant pas de cache, ce seuil passe à 2 Mo. Sur une machine avec des caches L2 de 6 Mo, les seuils augmentent de 50%.

Ces résultats montrent 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. Le déport de copie sur I/OAT n'utilise aucune ligne de cache contrairement à une copie directe de KNEM. La copie directe impose en effet deux accès au cache par le processus récepteur : les données à émettre doivent tout d'abord être lues en mémoire et placées dans les registres du processeur, elles doivent ensuite être écrites dans le tampon de réception, chaque opération impliquant un remplissage du cache. Pour éviter de polluer intégralement le cache local, celui-ci doit donc être au moins deux fois plus grand que la taille du message transféré. Même si les instructions *non-temporelles* des processeurs modernes permettent de réduire la pollution du cache lors des copies, il reste très préférable d'utiliser le moteur DMA d'I/OAT pour déporter les copies mémoires lors de grands messages. On peut estimer le seuil de taille de message à partir duquel utiliser I/OAT par :

$$DMA_{\min} = \frac{\text{taille du cache}}{2 \times \text{nombre de processus utilisant le cache}}$$

Lorsque deux processus partagent un cache L2 de 4 Mo, la formule nous ramène bien au seuil expérimental de 1 Mo. Lorsqu'aucun cache n'est partagé, chaque processus utilise son propre cache, le seuil passe alors à 2 Mo et vérifie ce rapport. En général les processus MPI sont distribués à raison d'un par cœur ce qui nous donne :

$$DMA_{\min} = \frac{\text{taille du cache}}{2 \times \text{nombre de cœurs partageant le cache}}$$

Le seuil où NEMESIS devrait passer du *double-buffering* classique (performant pour les petits messages) au LMT KNEM pourrait également être discuté. Tandis que le LMT `vmsplice` ne semble pas améliorer les performances avant 64 ko (seuil par défaut), KNEM devient lui intéressant à partir de 16 ko. Nous n'avons cependant pas trouvé de solution pour prédire ce seuil dynamiquement. Toutes ces performances seront détaillées dans la Section 4.

4. Évaluation des performances

Nous présentons une évaluation des performances de notre nouveau LMT KNEM avec MPICH2-Nemesis. La plupart de nos expériences ont été menées sur des architectures à deux processeurs quadri-cœurs

XEON INTEL E5345 (2,33 GHz, Figure 2). Chaque processeur possède deux caches L2 de 4 Mo partagés entre chaque paire de cœurs. Nous avons également effectué des tests sur un seul processeur quadri-cœurs XEON INTEL X5460 (3,16 GHz) disposant de deux caches L2 de 6 Mo et sur lequel nous avons observé des résultats similaires.

4.1. Communications point-à-point

Les figures 3 et 4 présentent le débit du test de *Pingpong* de la suite INTEL MPI benchmark (IMB) sur MPICH2 avec les différents LMTs. La réduction du nombre de copies induite par le LMT `vmsplice` est intéressante en comparaison au *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 dans le cache, la stratégie de double copie reste alors plus efficace. A première vue, un compromis permettant d'activer `vmsplice` dynamiquement lorsqu'aucun cache n'est partagé semble une solution prometteuse. Il est toutefois important de noter que ce résultat est spécifique au schéma de communication très simple d'un ping-pong. Comme nous le verrons en Section 4.2, les opérations collectives montrent un intérêt bien plus probant pour `vmsplice`. Comme attendu, KNEM améliore significativement les performances puisque il a été conçu pour des schémas d'accès spécifiques à MPI. En effet, `vmsplice` souffre par exemple de multiple appels système et de la division du transfert des données en segments de 64 ko.

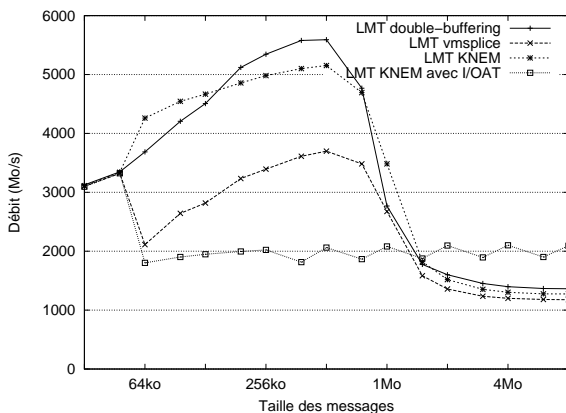


FIG. 3 – Pingpong IMB entre 2 processus partageant un cache L2 de 4 Mo.

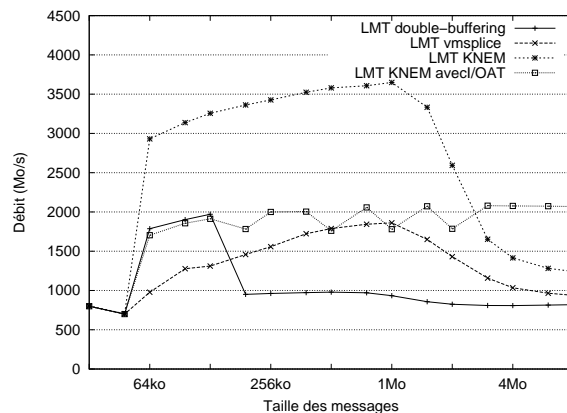


FIG. 4 – Pingpong IMB entre 2 processus ne partageant aucun cache.

Si aucun cache n'est partagé entre les cœurs, KNEM est plus de trois fois plus rapide que la copie *double-buffering* de NEMESIS et deux fois plus que `vmsplice`, atteignant 3,5 Go/s. Si les deux cœurs partagent un cache, les performances de KNEM s'avèrent sensiblement les mêmes que celles de NEMESIS. Nous avons estimé que KNEM devient intéressant lorsque la taille des messages atteint 8 ko, tandis que NEMESIS active normalement le LMT à partir de 64 ko.

L'utilisation du moteur I/OAT par KNEM est avantageuse pour des messages à partir de 1 Mo comme prédit par le calcul de seuil dynamique (Section 3.3). En effet, la soumission des copies à I/OAT nécessite un accès au dispositif matériel pour chaque morceau de mémoire physique contiguë. Cette initialisation le rend plus coûteux que l'utilisation de la stratégie KNEM sans I/OAT. Par contre, pour de très larges messages, I/OAT réduit la consommation processeur et la pollution de cache. Cela améliore considérablement les performances globales avec un facteur de 2,5 par rapport à NEMESIS. Les performances avec I/OAT sont légèrement variables à cause de problèmes d'alignement de zone mémoire que nous travaillons à résoudre.

Les processeurs XEON quadri-cœurs actuels permettent de placer deux processeurs sur la même puce sans partage de cache. Le comportement de notre stratégie dans ce cas est similaire au cas de non partage de cache observé précédemment (avec une légère dégradation de la bande passante globale due au

partage du lien mémoire). Cependant, nos processeurs sont un cas particulier conçu par INTEL pour offrir rapidement des processeurs quadri-cœurs à moindre coût, et les générations futures de processeurs devraient disposer de vrais caches partagés entre tous les cœurs, rendant cette situation caduque.

4.2. Opérations collectives

La figure 5 présente les performances de l'opération collective MPI *Alltoall* intra-nœud avec les LMTs *vmsplICE*, KNEM, et KNEM avec I/OAT. Nous avons observé des résultats similaires pour d'autres opérations collectives que nous ne présentons pas ici. Les résultats confirment que KNEM offre une large amélioration des performances en réduisant la consommation processeur et la pollution de cache. La bande passante obtenue est ainsi cinq fois plus élevée pour les messages de taille moyenne (autour de 32 ko) et deux fois plus élevée pour de très grands messages (plusieurs megaoctets) grâce au déport de copie mémoire sur le matériel I/OAT. Notons que *vmsplICE* offre, grâce à la réduction de la pollution

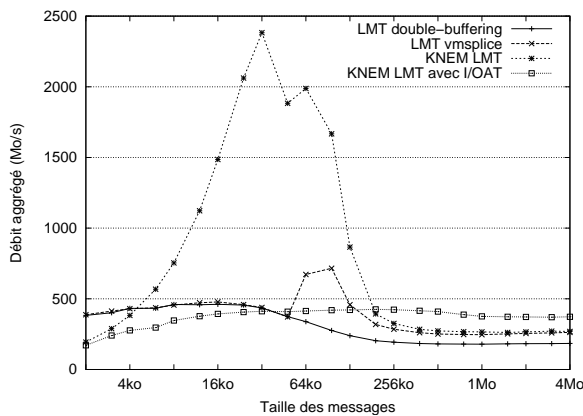


FIG. 5 – Débit (agrégé) d'un Alltoall IMB entre 8 processus locaux.

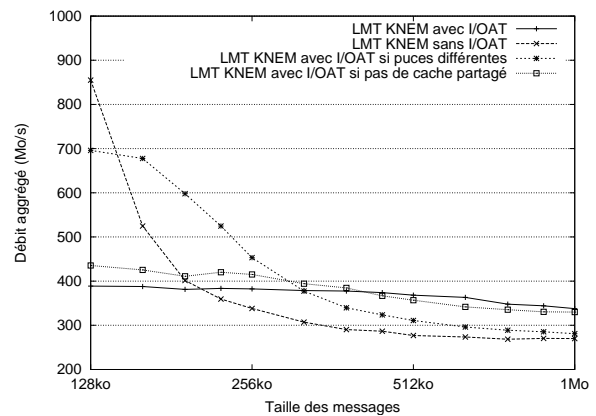


FIG. 6 – Débit (agrégé) d'un Alltoall IMB (8 processeurs) : mélange des différentes stratégies de KNEM.

de cache et de l'utilisation CPU, une légère amélioration des performances de NEMESIS, appréciable pour les utilisateurs qui ne peuvent pas se permettre le chargement du module noyau dédié KNEM. L'utilisation de I/OAT devient intéressante autour de 200 ko, soit bien moins que le seuil présenté en Section 3.3. Cela est dû au fait que huit processus interviennent dans le test *Alltoall*, tandis que deux intervenaient dans le test *Pingpong*. Avec un nombre de messages transférés simultanément plus élevé, la saturation des caches et bus mémoire est plus importante. Il devient donc intéressant d'utiliser I/OAT plus tôt. Les opérations collectives semblent également souffrir très tôt lorsque le *double-buffering* de NEMESIS est utilisé. KNEM est ainsi avantageux pour des messages de 4 ko alors que NEMESIS active le LMT après 64 ko. Nous envisageons donc de réduire de manière dynamique la valeur de ce seuil.

De plus, la figure 6 montre que l'utilisation conjointe de KNEM avec et sans I/OAT semble améliorer plus en avant les performances des opérations collectives autour du seuil. L'emploi partiel de I/OAT décharge en effet les caches d'un certain nombre de données, permettant à la copie directe d'être performante plus longtemps. Confier à I/OAT les transferts entre des processus placés sur des puces différentes ou sur des cœurs ne partageant pas de cache offre ainsi une augmentation de débit autour du seuil proche de 20%. D'un point de vu général, on constate que plus les processus sont éloignés hiérarchiquement plus il devient intéressant d'utiliser I/OAT tôt. Notons que les idées moins logiques où on utiliserait I/OAT uniquement lorsque les processus partagent un cache améliorent tout de même le débit, probablement grâce à une réduction des contentions mémoire.

4.3. NAS Parallel Benchmarks

La figure 7 présente les temps d'exécution de différents tests parallèles de la suite NAS. La plupart de ces tests n'utilisent que très peu de gros messages, et ne montrent ainsi que de légères variations de

² Cette absence est due à un bogue connu mais encore non résolu dans NEMESIS.

Test NAS	LMT double-buffering	LMT vmssplice	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%

FIG. 7 – Temps d'exécution de différents tests parallèles de la suite NAS.

	LMT double-buffering	LMT vmssplice	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

FIG. 8 – Défauts de cache L2 pendant l'exécution de différentes applications. IS et Alltoall utilisent les 8 cœurs de la machine. Les processus des Pingpong sont placés sur des puces différentes.

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%. Pour expliquer ce gain pour le test IS, nous présentons en Figure 8 le nombre de défauts de cache³ produits lors de l'exécution, mesuré à l'aide de PAPI [2]. Elle indique que le temps d'exécution de IS est en quelque sorte linéaire avec le nombre de défauts de cache. En utilisant une copie directe avec KNEM, ou mieux avec KNEM et I/OAT, la pollution de cache est réduite. En conséquence le nombre de défauts de cache diminue également, d'où une réduction du temps d'exécution.

L'observation des comportements point-à-point et collectif, confirme que KNEM évite un nombre de défauts de cache significatif pour les grands messages et des opérations collectives, tandis que l'implémentation de la copie *double-buffering* de NEMESIS ne reste compétitive que pour de petits messages.

5. Conclusions et perspectives

Nous avons présenté une nouvelle technique pour le passage de messages intra-nœud dans le contexte de MPICH2, une implémentation portable haute-performance du standard MPI. La stratégie habituelle de communication basée sur une double copie à travers un tampon mémoire partagé présente de fortes limites en performance en raison de la grande consommation de temps processeur et de la pollution de cache qu'elle induit. Notre nouvelle stratégie de communication pour les grands messages est basée sur le nouveau module noyau dédié KNEM. Ce modèle offre une implémentation conçue pour MPI et permet de profiter des fonctionnalités de déport de copie mémoire grâce au matériel I/OAT d'INTEL qui est désormais très largement diffusé.

L'évaluation de ces méthodes montre que les performances de MPICH2 bénéficient largement de nos travaux. La bande passante des transferts est significativement accrue, même en l'absence de cache partagé entre deux cœurs. De plus, elle se maintient à plus de 2 Go/s même pour de très larges messages grâce à l'utilisation de la technologie I/OAT. Notre étude des opérations collectives révèle que la réduction du nombre de copies mémoire pour le transfert de grands messages permet de profiter largement d'une diminution de l'utilisation du processeur et d'une meilleure utilisation du cache. Le test NAS IS montre ainsi un nombre restreint de défauts de cache et une accélération de 25% grâce à KNEM.

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.

Nous avons également souligné l'importance de calculer dynamiquement les seuils pour obtenir une sélection optimale des techniques de transfert. Aucune méthode n'est optimale en toutes situations. Une approche associant les différentes stratégies est indispensable pour obtenir de bonnes performances dans les différents benchmarks et applications. Nous envisageons d'améliorer nos heuristiques de prédiction des seuils. Les seuils pourraient par exemple être calibrés spécifiquement pour les opérations collectives grâce à l'assistance des couches supérieures de MPICH2 qui savent si plusieurs transferts de larges messages sont initialisés en parallèles. Les caractéristiques des caches plus petits pourraient également être pris en compte pour la recherche des seuils entre la double-copie NEMESIS et le LMT. L'augmentation du nombre de cœurs et de larges caches partagés dans les processeurs très récents tels que le NEHALEM INTEL, ainsi que la démocratisation des architectures à accès mémoire non uniforme (NUMA) maintiennent le besoin d'ajuster soigneusement les communications intra-nœud en fonction des affinités des processus. Nous prévoyons d'améliorer notre modèle pour exploiter automatiquement ces nouvelles architectures en choisissant les stratégies et seuils les plus adaptés. Nous prévoyons aussi d'améliorer KNEM en activant plus de recouvrement des communications et une meilleure réactivité avec une interface noyau plus flexible. L'idée de faire descendre les informations sur les opérations collectives dans le LMT (voire dans le module KNEM) doit être étudiée car elle pourrait entraîner une manipulation des données coûteuse car effectuée une seule fois par opération et non pas par message.

Bibliographie

1. « The splice () weekly news », avril 2006. <http://lwn.net/Articles/181169/>.
2. S. BROWNE, J. DONGARRA, N. GARNER, G. HO et P. MUCCI. « A Portable Programming Interface for Performance Evaluation on Modern Processors ». *The International Journal of High Performance Computing Applications*, 14(3) :189–204, 2000.
3. D. BUNTINAS, G. MERCIER et W. GROPP. « Data Transfers between Processes in an SMP System : Performance Study and Application to MPI ». *ICPP 2006. International Conference on Parallel Processing*, pages 487–496, août 2006.
4. D. BUNTINAS, G. MERCIER et W. GROPP. « Design and Evaluation of Nemesis, a Scalable Low-Latency Message-Passing Communication Subsystem ». Dans Stephen John TURNER, Bu Sung LEE et Wentong CAI, éditeurs, *Proceedings of the 6th IEEE International Symposium on Cluster Computing and the Grid (CCGRID '06)*, pages 521–530, Washington, DC, USA, mai 2006. IEEE Computer Society.
5. D. BUNTINAS, G. MERCIER et W. GROPP. « Implementation and Shared-Memory Evaluation of MPICH2 over the Nemesis Communication Subsystem ». Dans *Recent Advances in Parallel Virtual Machine and Message Passing Interface : Proc. 13th European PVM/MPI Users Group Meeting*, Bonn, Germany, septembre 2006.
6. Brice GOGLIN. « High Throughput Intra-Node MPI Communication with Open-MX ». Dans *Proceedings of the 17th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP2009)*, Weimar, Germany, février 2009. IEEE Computer Society Press.
7. RL. GRAHAM, TS. WOODALL et JM. SQUYRES. « Open MPI : A Flexible High Performance MPI ». Dans *Proceedings, 6th Annual International Conference on Parallel Processing and Applied Mathematics*, Poznan, Poland, septembre 2005.
8. A. GROVER et C. LEECH. « Accelerating Network Receive Processing (Intel I/O Acceleration Technology) ». Dans *Proceedings of the Linux Symposium*, pages 281–288, Ottawa, Canada, juillet 2005.
9. HW. JIN, S. SUR, L. CHAI et DK. PANDA. « Lightweight Kernel-Level Primitives for High-Performance MPI Intra-Node Communication over Multi-Core Systems ». Dans *Proceedings of the IEEE International Conference on Cluster Computing (Cluster'07)*, Austin, TX, septembre 2007.
10. MESSAGE PASSING INTERFACE FORUM. « MPI : A Message-Passing Interface Standard, Version 2.1 », June 2008. <http://www.mpi-forum.org/docs/mpi21-report.pdf>.
11. 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>.
12. NETWORK-BASED COMPUTING LAB, THE OHIO STATE UNIVERSITY. « MVAPICH : MPI for InfiniBand over VAPI Layer ». <http://nowlab.cse.ohio-state.edu/projects/mpi-iba/>.
13. Bernard Tourancheau PARTICK GEOFFRAY, Loïc Prylli. « BIP-SMP : High Performance Message Passing over a Cluster of Commodity SMPs ». Dans *Proceedings of the 1999 ACM/IEEE conference on Supercomputing*, Portland, Oregon, United States, 1999.