NewMadeleine : ordonnancement et optimisation de schemas de communication haute performance.

Elisabeth Brunet, Olivier Aumage, Raymond Namyst

NewMadeleine : ordonnancement et optimisation de schémas de communication haute performance

Élisabeth BRUNET* — **Olivier AUMAGE*** — **Raymond NAMYST***

* *Projet INRIA* RUNTIME*, LaBRI*
Université Bordeaux I - 351, cours de la Libération - 33405 Talence Cedex
{elisabeth.brunet, olivier.aumage, raymond.namyst}@labri.fr

RÉSUMÉ. *Malgré les progrès spectaculaires accomplis par les interfaces de communication pour réseaux rapides ces quinze dernières années, de nombreuses optimisations potentielles échappent encore aux bibliothèques de communication. La faute en revient principalement à une conception focalisée sur la réduction à l'extrême du chemin critique afin de minimiser la latence. Dans cet article, nous présentons une nouvelle architecture de bibliothèque de communication bâtie autour d'un puissant moteur d'optimisation des transferts dont l'activité s'accorde avec celle des cartes réseau. Le code des stratégies d'optimisations est générique et portable, et il est paramétré à l'exécution par les capacités des pilotes réseau sous-jacents. La base de données des stratégies d'optimisation prédéfinies est facilement extensible. L'ordonnanceur est en outre capable de mixer de façon globalisée de multiples flux logiques sur une ou plusieurs cartes physiques, potentiellement de technologies différentes en multi-rail hétérogène.*

ABSTRACT. *Despite the spectacular progress made in the field of communication interfaces for fast networks over the last fifteen years, many potential optimizations are still unexplored by nowadays communication libraries. The cause of this situation resides in a design mostly focused on shortening the critical path to the maximum, in order to minimize the latency. In this paper, we present a new communication library architecture built around a powerful transfer optimizing engine, which coordinates its own activity with that of the network cards. The source code of the optimization strategies is generic and portable, and is parameterized at runtime by the capacities of the underlying network drivers. The database of predefined optimization strategies can easily be extended. The scheduler is also able to mix multiple logical communication flows in a global manner onto one or more physical network cards, potentially of different technologies in a heterogeneous multi-rail.*

MOTS-CLÉS : *Communication Haute Performance, Ordonnancement, Optimisation.*

1re soumission à *Technique et Science Informatique*, le 18 décembre 2006

KEYWORDS: *High Performance Communication, Scheduling, Optimization.*

1. Introduction

Dans le paysage du calcul parallèle, la percée des machines que sont les grappes de PC repose évidemment beaucoup sur leur avantageux rapport performance/coût, mais aussi sur les progrès réalisés en matière de réseaux d'interconnexion *rapides*. Et dans ce domaine, une fois n'est pas coutume, les évolutions du matériel (latences très faibles associées à des débits élevés) ont été accompagnées par des progrès logiciels remarquables. Les recherches menées depuis une quinzaine d'années en matière de bibliothèques de communication ont permis, en associant un vaste éventail de techniques telles que les communications en mode utilisateur, les échanges zéro-copie, le déport de certaines opérations sur les cartes réseaux ou encore l'optimisation de pipelines logiciels, de réduire au minimum le chemin critique des traitements réalisés lors d'une communication. Aujourd'hui, le surcoût logiciel occasionné par les bibliothèques spécialisées (ELAN/QUADRICS (Quadrics Ltd., 2003) ou encore MX/MYRINET (Myricom, Inc., 2003)) se réduit à quelques centaines de cycles-processeur par transaction. Mieux encore, les implémentations récentes du standard MPI ((Buntinas *et al.*, 2006), (Graham *et al.*, 2005)), qui sont constituées pour une grande part d'appels directs aux routines de l'interface sous-jacente, affichent quasiment les mêmes performances.

Pour des applications n'utilisant que des requêtes simples d'un seul fragment et des schémas de communication très réguliers, l'utilisation de tels supports de communication est idéal. Les applications actuelles tendent cependant à se complexifier du point de vue des communications. Les applications deviennent plus modulaires et plus composites : chaque module ou composant génère plusieurs flots de communication à multiplexer. La glue logicielle et les environnements de contrôle génèrent également leurs propres flots et mettent en œuvre des protocoles élaborés de type RPC ou RMI : ces requêtes sont composées de fragments multiples (numéro de service ou méthode, arguments, données) et ces fragments se structurent eux-mêmes en messages, par le truchement de relations de dépendance internes. Pour le traitement de telles requêtes, le matériel de communication pourrait être bien mieux exploité qu'il ne l'est actuellement. C'est ce que nous démontrons dans cet article où nous introduisons notre propre interface de communication **NewMadeleine** destinée à pallier ce problème tout en restant efficace dans le traitement des requêtes simples. Nous en exposons par ailleurs les éléments clés d'implémentation et les premiers résultats d'évaluation obtenus. Nous commentons enfin les principaux travaux apparentés du domaine au regard de notre contribution.

2. Limitations des interfaces de communication actuelles

Tout d'abord, les besoins des applications de calcul hautes performances ne se réduisent pas toujours à minimiser la latence des échanges de données pris de manière individuelle. En effet, privilégier la bande passante ou encore le recouvrement des temps de communication peut s'avérer judicieux pour des applications utilisant un système de stockage de données à distance ou encore des applications ayant une

forte charge de calcul à exécuter. De plus, nous avons montré par le passé (Aumage *et al.*, 2002) que les applications et environnements de programmation mettant en œuvre des protocoles ou des mécanismes de communication de haut niveau (RPC, Java RMI, CORBA, et bien d'autres) peuvent tirer un grand bénéfice d'une interface de communication plus puissante que MPI. Le manque d'expressivité de cette dernière interface, même si elle se comporte de manière optimale sur les incontournables tests de type *ping-pong*, ne permet pas d'exprimer des indications subtiles telles que les dépendances entre les différents fragments (service, arguments, objet concerné) d'une invocation de méthode distante. C'est pourtant en permettant l'expression et en exploitant l'information de ces contraintes qu'un support de communication tirer parti des capacités du réseau de manière optimale : il peut alors choisir d'accumuler des paquets afin d'utiliser des fonctionnalités de collection/dissémination (*gather*/*scatter*) ou agréger plusieurs petites requêtes en une seule plus grosse si les contraintes sont suffisamment relâchées pour le permettre, réordonner des paquets, ou encore favoriser l'acheminement prioritaire de fragments importants (comme un numéro de service RPC, nécessaire pour préparer de façon anticipée les zones de mémoire destinées à recevoir les arguments du service). On le voit, pour ces classes d'applications caractérisées par des schémas de communications complexes et irréguliers, il ne s'agit pas de faire une simple traduction des opérations de communication en appels à des routines du pilote réseau, mais bien d'*interpréter*, *réorganiser* et *optimiser* le flux d'opérations de communication de façon dynamique.

Par ailleurs, l'évolution des techniques de programmation vers des applications modulaires et composites, résultant de l'intégration de divers environnements de programmation (cf *PadicoTM* (Denis *et al.*, 2003)) et souvent multiprogrammées pour exploiter les architectures multiprocesseurs modernes (multicores), engendrent des besoins applicatifs nouveaux : cette évolution amène une multiplication des flux de communication logiques entre les différents modules de deux processus communiquant. Les capacités de multiplexage physique des cartes réseau — très réduites en raison de la consommation de ressources occasionnée dans la mémoire embarquée sur les cartes — ne sont pas prévues pour répondre à cette demande (seulement 3 unités de multiplexage par défaut pour une carte de communication Myrinet avec MX, par exemple). Il est donc nécessaire de « mixer » les multiples flux logiques sur les ressources de multiplexage physiques. Pourquoi ne pas en profiter pour appliquer les techniques d'optimisation évoquées ci-dessus (agrégation, réordonnancement) de façon globale à l'ensemble des flux plutôt qu'indépendamment sur chaque flux pris séparément? Une telle approche permettrait d'appliquer de façon circonstancielle des optimisations purement opportunistes, à condition que la bibliothèque de communication soit capable de gérer des flux de données non-déterministes du côté du récepteur, ce qui requiert l'utilisation de techniques non triviales.

La section suivante expose les techniques que nous proposons pour exploiter ces conclusions et présente l'architecture qui en dérive.

3. Proposition

NEWMADELEINE n'est pas une simple évolution de l'interface de communication MADELEINE mais une nouvelle version à part entière. Son architecture totalement innovante repose sur différents concepts exposés dans les sous-sections 3.1 et 3.2 suivantes.

3.1. *Constitution d'une fenêtre de travail*

Les optimisations dynamiques multi-paquets et multi-flux mentionnées précédemment nécessitent par définition une *fenêtre* de travail de plusieurs paquets pour être réalisables. Or, le fonctionnement traditionnellement *synchrone* des supports de communication — où les opérations de communications sont corrélées à l'activité de l'application — se prête mal à ce mode opératoire : soit un paquet est immédiatement transmis sur le réseau lorsqu'il est soumis par l'application et il n'y a pas d'accumulation (c'est le fonctionnement habituel), soit le support de communication devrait garder arbitrairement le paquet jusqu'à ce que l'application envoie un ou plusieurs autres paquets afin de constituer la fenêtre, ce qui n'est pas réaliste du point de vue de la latence induite et qui n'est de toute façon pas acceptable d'un point de vue algorithmique en raison des risques de deadlock. Afin de permettre la constitution effective de la fenêtre recherchée sans subir de tels désagréments, il est nécessaire d'abandonner la corrélation entre l'application et le traitement des opérations de communication. Dans ce cas, les opérations doivent être traités suivant l'activité de la carte (ou des cartes) de communication. Tant que celle-ci est occupée, l'ordonnanceur accumule simplement une fenêtre de paquets. Lorsque celle-ci devient inoccupée, il analyse la fenêtre de paquets collectés entre-temps et il en extrait une requête qu'il soumet au réseau pour redonner du travail à la carte. La constitution de la fenêtre est donc naturelle, et ne présente pas les inconvénients algorithmiques de l'approche synchrone.

3.2. *Stratégies, tactiques et sélection d'optimisation*

La fenêtre de travail étant constituée, se pose maintenant la question du travail d'optimisation proprement dit. Considérons l'état des unités de multiplexages physiques à un instant donné. Si au moins une unité de multiplexage est disponible à cet instant, il faut lui donner du travail à réaliser en appliquant une *fonction d'optimisation* chargée de sélectionner (ou de générer, par exemple, par une agrégation) le prochain paquet à soumettre à chacune des unités inactives en considérant les paquets de la fenêtre de travail. En entrée de la fonction d'optimisation, nous avons une grande variété d'arguments potentiels, dont voici une liste non exhaustive : le nombre de paquets dans la fenêtre, leurs caractéristiques respectives (destination, flux, longueur, numéro de séquence, attributs de dépendance), les caractéristiques nominales et fonctionnelles du réseau, éventuellement des indications de l'application sur la po-

litique d'ordonnancement des paquets, ainsi que le nombre d'unités de multiplexages physiques disponibles et leur état d'activité à l'instant considéré.

Devant un tel nombre de paramètres, plusieurs stratégies d'optimisations peuvent être payantes. Plutôt que d'imposer une improbable fonction d'optimisation ultime construite sur une stratégie figée, nous proposons au contraire que la fonction d'optimisation soit sélectionnable (à terme dynamiquement) parmi un ensemble de stratégies extensible et programmable : chaque *stratégie* combine l'application d'un certain nombre de *tactiques* d'optimisation. Chaque tactique est elle-même construite à partir d'opérations élémentaires de manipulation de paquets du panel d'opérations usuelles, dans la réalisation d'un objectif d'optimisation particulier.

Si aucune unité de multiplexage n'est disponible, le cas est simple : on remet le travail d'optimisation à une date ultérieure. Une autre possibilité envisageable serait de préparer un unique paquet à émettre pour anticiper sur la prochaine libération d'une des unités, et pouvoir la réalimenter immédiatement (donc en travaillant avec un coup d'avance). Une troisième alternative serait d'appliquer malgré tout la fonction d'optimisation si la quantité de paquets accumulés atteint un certain seuil.

3.3. *Architecture*

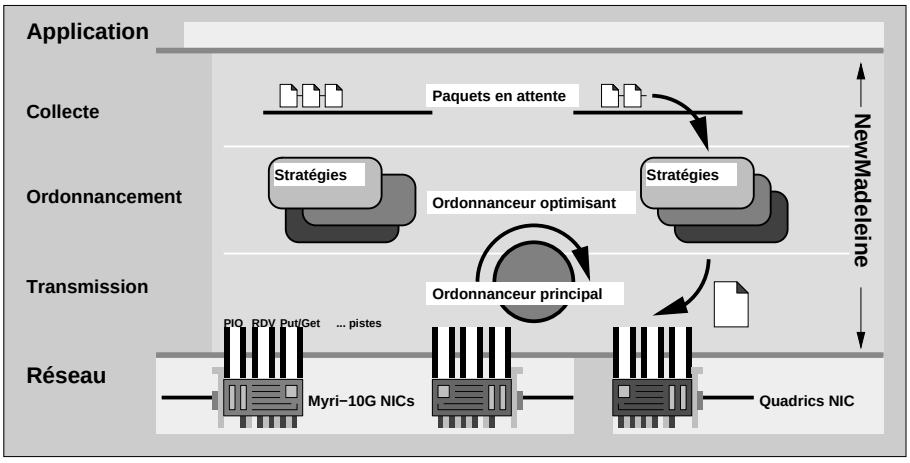


Figure 1. *Architecture de NewMadeleine*

L'architecture de NEWMADELEINE est organisée en trois couches : une couche de collecte des données auprès des applications, une couche d'ordonnancement et d'optimisation et une couche de transfert pilotant les cartes de communication (voir la figure 1).

La couche de transfert mime les cartes réseaux à la façon d'un ordonnanceur de processus qui, lorsqu'il est invoqué par un processeur, déroule un algorithme permet-

tant de choisir un nouveau processus prêt. Son rôle est de surveiller l'état d'avancement des cartes, et de dialoguer avec la partie supérieure pour obtenir de nouveaux paquets optimisés dès que cette dernière achève sa tâche. Pour cela, une partie générique interroge successivement les pilotes spécifiques à chaque réseau, uniques détenteurs des primitives permettant l'interrogation de la carte elle-même. C'est une couche très fine sans intelligence : les primitives des pilotes se résumant à une primitive d'envoi, de réception et de scrutation.

La couche d'ordonnancement-optimisation est la partie intermédiaire. Les paquets déposés par l'application sont tirés par les cartes réseau en temps voulu par son intermédiaire. Elle est alors chargée de manipuler la liste pour former un paquet et fournir un assemblage à la fois conforme aux contraintes applicatives et avantageux du point de vue des performances. Ce paquet est alors directement transmis à la carte demandeuse pour émission immédiate. Pour ne pas faire patienter la carte, on pourrait préparer quelques paquets en avance. Cependant, il est préférable de se limiter à un nombre restreint de tels paquets car cela laisse à l'ordonnanceur une marge de manœuvre plus importante qui ne peut être que bénéfique. Finalement, la carte ne pouvant consommer qu'un paquet à la fois, un paquet en cours et un paquet d'avance est une mesure adaptée.

Cette couche met également en œuvre les protocoles réseaux basiques nécessaires au bon déroulement de communication (tels que le contrôle de flux, les prises de rendez-vous, etc.) mais assure aussi la remontée du succès des requêtes à l'application concernée.

La couche de collecte est chargée de recenser les données déposées par les différents flux de communication ainsi que les meta-données nécessaires à leur identification par le récepteur (i.e. numéro de canal, identifiant de l'expéditeur, numéro de séquence). Une fois encapsulées et afin de permettre un équilibrage de charge entre les différentes cartes présentes, les données collectées sont alors insérées sur une liste dédiée à la technologie réseau désignée par l'application ou, à défaut, sur une liste générale pour un équilibrage de charge sur l'ensemble des cartes (même de technologie différente) en présence.

4. Implantation

Nous détaillons maintenant les principaux aspects d'implantation propres à NEWMADELEINE. Pour cela, nous étudions séparément chacune de ces trois couches.

4.1. *La couche de transfert*

Afin de pouvoir intégrer facilement de nouvelles technologies réseaux, nous avons défini une interface minimale — commune aux différents pilotes — qui se résume à des fonctions d'initialisation, de fermeture, d'envoi, de réception et de scrutation. Ces

fonctions sont de très bas niveau puisque leur corps ne se restreint pratiquement qu'à l'appel direct de la primitive de l'interface de communication bas niveau associée.

Afin de permettre à l'ordonnanceur de cibler au mieux chaque technologie réseau, un ensemble de caractéristiques qualitatives et quantitatives est consigné dans un tableau de *capacités*. Parmi ces capacités se trouvent des informations telles que le support de requêtes non contigües, le support des accès à la mémoire distante ou encore le seuil de taille au-delà duquel, une requête doit faire l'objet d'un transfert synchrone avec rendez-vous préalable.

NEWMADELEINE a actuellement été portée au dessus de GM/MYRINET, MX/MYRINET, ELAN/QUADRICS, SISCI/SCI et TCP/ETHERNET et est en cours d'adaptation sur INFINIBAND.

4.2. *La couche d'ordonnancement-optimisation*

Le développement de stratégies d'optimisation étant à la fois délicat et en général commun à de nombreuses technologies réseaux, nous avons élaboré un cadre facilitant l'écriture de *stratégies* d'ordonnancement indépendamment du ou des réseaux physiques sous-jacents. Ce cadre est le réceptacle d'une bibliothèque de stratégies, destinée à être enrichie au gré des développements et expérimentation. L'ajout d'une stratégie à la bibliothèque permet de faire bénéficier chacune des technologies réseau de cette stratégie. De plus, l'étude comparative de stratégies différentes s'en trouve grandement facilitée.

Actuellement, deux stratégies sont disponibles : une stratégie d'agrégation et une stratégie multi-rail. La stratégie d'agrégation consiste à agréger les paquets soumis par l'application tant que leur taille cumulée ne requiert pas un envoi via un rendez-vous préalable. La stratégie multi-rail a comme but d'exploiter simultanément et surtout *efficacement* toutes les cartes réseaux disponibles. Pour cela, elle distribue les messages courts sur le réseau le plus rapide. Elle découpe les plus gros de manière dynamique si nécessaire, en pondérant la taille des fragments selon les performances relatives des différents réseaux.

4.3. *La couche de collecte*

NEWMADELEINE fournit différentes interfaces utilisateur. La première est une interface de type *échange de message* qui permet en outre une construction *incrémentale* des messages. Chaque message peut ainsi être composé de plusieurs fragments de données éparses en mémoire. Les frontières de messages sont marquées par une barrière de synchronisation. Cette interface est similaire à l'interface traditionnelle proposée dans les versions précédentes de MADELEINE.

Afin de pouvoir nous positionner par rapport aux implémentations de MPI, nous avons également développé une seconde interface utilisateur, ce qui se réalise aisé-

ment étant donné l'architecture modulaire de NEWMADELEINE. Cette seconde interface, nommée MAD-MPI, implémente un sous-ensemble de primitives de MPI. Elle se restreint essentiellement aux opérations de bases telles que les opérations de communication point-à-point non bloquantes `MPI_Isend` et `MPI_Irecv` et les primitives de complétion `MPI_Test` et `MPI_Wait`. Elle propose en revanche un support optimisé pour les types dérivés (Furmento *et al.*, 2005). Chaque type dérivé décrit un ensemble de blocs de données non contigus en mémoire. NEWMADELEINE se charge donc de réordonner les fragments de messages constitués par ces blocs et de les combiner afin de générer des requêtes maximisant l'efficacité de l'utilisation des réseaux sous-jacents.

5. Évaluation

L'objectif de cette section est dans un premier temps de mettre en évidence le gain qu'il peut y avoir à optimiser un schéma de communication par rapport à le calquer à celui décrit par l'application. Pour cela, nous procédons à différentes évaluations qui mettent en avant les stratégies actuellement implémentées dans NEWMADELEINE : la stratégie d'agrégation et la stratégie multi-rail. Dans un second temps, nous comparerons MAD-MPI– notre sous-ensemble de primitives de MPI – aux versions les plus répandues d'implémentation de MPI : MPICH et OPEN MPI.

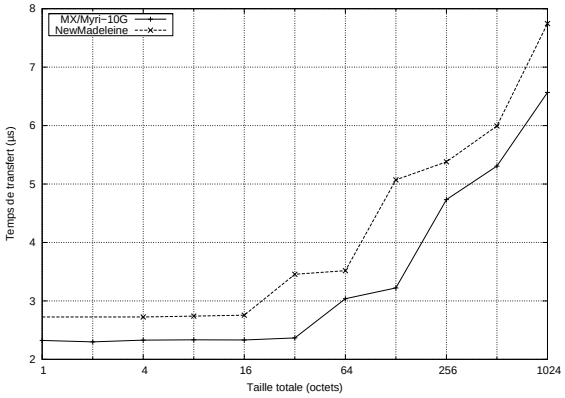
5.1. *Plate-forme d'évaluation.*

L'ensemble des expérimentations ont été menées sur une grappe d'OPTERON dual-core 1.8 GHz équipés de 1 Mo de cache L2 et de 1 Go de mémoire vive sous LINUX VERSION 2.6.17. Les nœuds sont interconnectés par deux réseaux haute performance : MYRI-10G avec le pilote MX1.2.0 et QUADRICS QM500 avec le pilote ELAN.

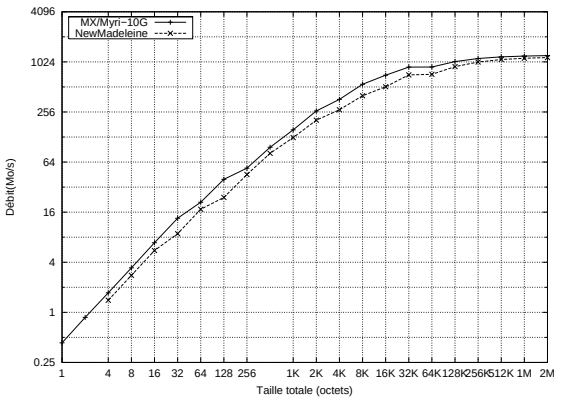
5.2. *Surcoût brut par message élémentaire.*

Afin de nous donner des points de référence, nous commençons par mesurer à l'aide du test classique du ping-pong les performances brutes de NEWMADELEINE en les comparant à celles des bibliothèques de communication MX (figure 2) et ELAN (figure 3).

Dans les deux cas, nous observons un surcoût logiciel de moins de 0.5 μs de la part de notre bibliothèque. Deux facteurs sont notamment en cause. Tout d'abord, les données sont systématiquement accompagnées d'entêtes afin de permettre les inversions et le multiplexage. Ainsi, à taille de données utiles égales, la masse (entêtes + données utiles) des données effectivement transmise est supérieure avec NEWMADELEINE. Ensuite, l'application d'optimisations sur une liste d'un élément unique ne



(a) Latence.



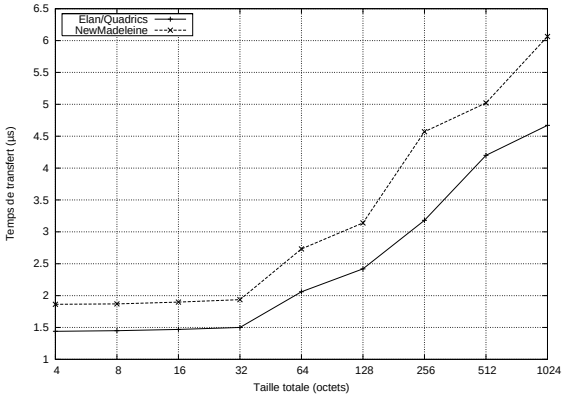
(b) Débit.

Figure 2. *Ping-pong* - NEWMADELEINE *vs* MX-10G.

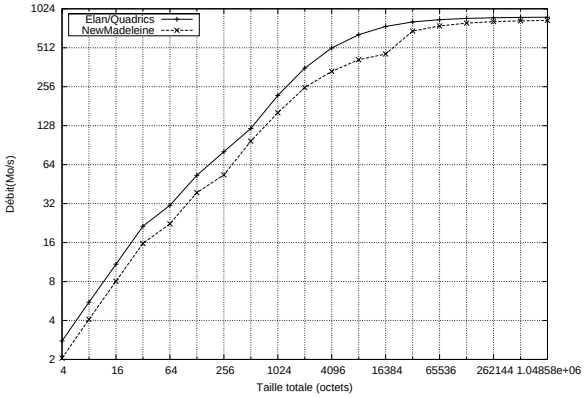
peut être que superflue par rapport à un envoi direct. Ceci pourra par la suite se simplifier par le court-circuitage de l'optimiseur dans une telle situation. Le débit atteint 1170 Mo/s sur MYRI-10G et 837 Mo/s sur QUADRICS soit finalement une perte de moins de 5%. Étant donné qu'aucune optimisation ne peut être apportée à un échange de données d'un seul élément, ce type d'évaluation n'est pas du tout un cas favorable à nos politiques d'optimisation. Ainsi, nous estimons que les performances obtenues sont satisfaisantes.

5.3. *Ping-pong multi-segment.*

À présent, nous nous intéressons à un cas de figure où l'agrégation de communication est favorable. Pour cela, nous considérons un ping-pong où chaque séquence



(a) Latence.

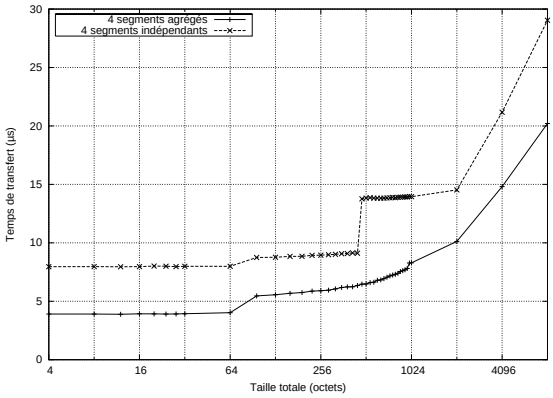


(b) Débit.

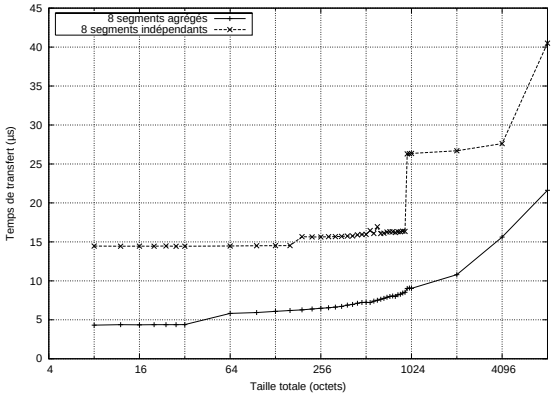
Figure 3. *Ping-pong* - NEWMADELEINE *vs* QUADRICS.

d'envoi est composée d'une multitude d'envois de segments de même petite taille, ceci permettant d'accroître considérablement nos opportunités d'agrégation et donc d'optimisation. Nous menons l'expérience une fois en autorisant l'agrégation de ces requêtes et une fois en l'interdisant. Les performances atteintes sont présentées dans les figures 4 et 5.

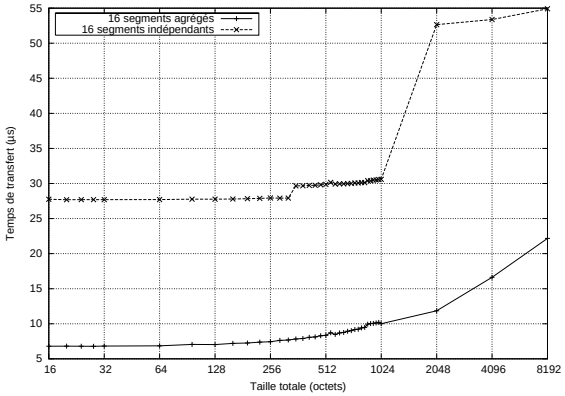
Il y a un gain certain à agglomérer le plus de segments possibles puisque plus on agglomère plus l'écart se creuse entre les versions avec et sans agrégation que ce soit sur MYRY-10G ou QUADRICS. Cette optimisation est ainsi particulièrement adaptée aux applications qui s'échangent très fréquemment des messages courts comme des messages de contrôle ou de synchronisation.



(a) 4 segments sur MYRI-10G

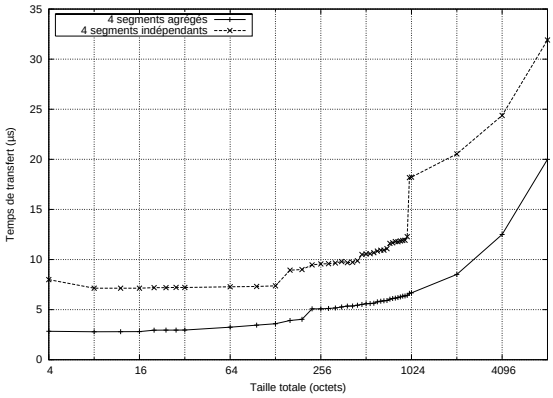


(b) 8 segments sur MYRI-10G

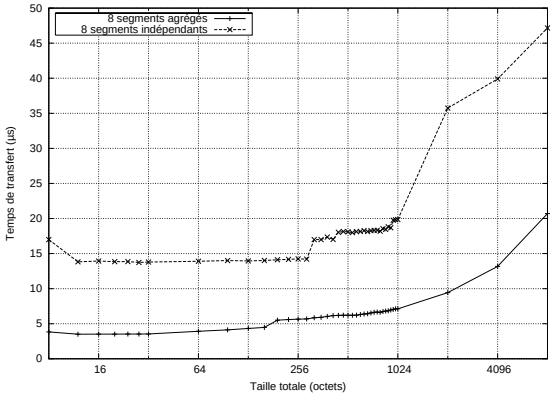


(c) 16 segments sur MYRI-10G

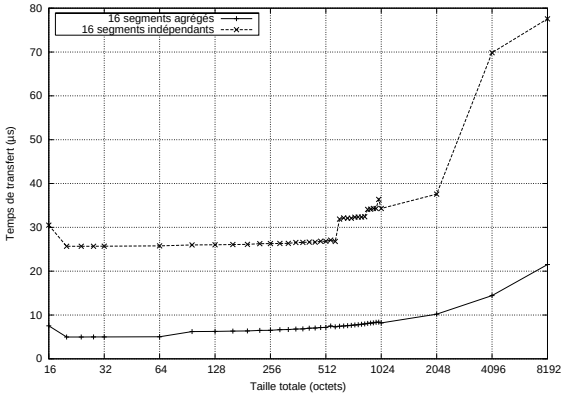
Figure 4. *Ping-pong multi-segment sur* MYRI-10G.



(a) 4 segments sur QUADRICS



(b) 8 segments sur QUADRICS



(c) 16 segments sur QUADRICS

Figure 5. *Ping-pong multi-segment sur* QUADRICS.

5.4. *Demandes de pages auprès d'une MVP.*

Nous mettons à présent l'accent sur l'inversion des messages. Pour cela, nous choisissons de mettre en œuvre une pseudo mémoire virtuellement partagée (MVP) afin de mélanger des données dont la taille nécessite des envois par une méthode de transfert différente. En effet, des demandes de pages consécutives offrent l'opportunités d'agréger les arguments de toutes les demandes de pages et leur envoi avant même d'envoyer la première page demandée si l'acquittement associé n'a toujours pas été reçu. Un algorithme schématique de demandes de pages auprès d'une MVP est décrit à la figure 6. Le schéma de communication produit par une interface de communication ne faisant pas de réordonnancement est présenté à la figure 7(a). Chaque demande d'envoi (traduite par un appel à la fonction *pack* (voir la figure 6)) se solde par l'envoi effectif demandé et symétriquement en réception (via des appels à la fonction *unpack*). A la sortie de l'appel, la requête est terminée du point de vue du programmeur. Les paquets n'ont pourtant pas de dépendances bloquantes entre eux à l'exception de la réception notée (7) du côté client qui dépend de la réception précédente (6) dans la figure 6 et cela au sein d'une seule et même demande. Dans notre proposition, nous donnons à l'application la possibilité d'exprimer ces dépendances afin d'autoriser des inversions et de mieux ordonnancer les paquets en exploitant le réseau sous-jacent.

Ainsi, si on opte pour une tactique agrégeant les paquets jusqu'à un certain seuil, on pourra observer de nouveaux schémas de communications tels que ceux décrits dans les figures 7 (b) et 7 (c). Les capacités même du réseau sous-jacent à, entre autres, gérer ou non les techniques du collection/dissémination sont un point déterminant dans la façon dont l'ordonnancement va se dérouler.

Nous présentons dans le tableau 8 une moyenne des résultats obtenus pour la demande et la réception de trois pages de 65 Ko. Seule l'évaluation au dessus de MYRI-10G utilise la fonctionnalité de *gather/scatter* puisque ELAN ne la propose pas. La taille des pages a été choisie de manière à ce que leur transfert ne soit pas fait de la même manière que la requête elle-même (on force le passage par un rendez-vous).

Finalement, alors que le test ne porte que sur un nombre réduit d'échanges, l'écart entre les résultats est d'ores et déjà important. Ceci est particulièrement intéressant pour toutes les applications basées sur les schémas de communication évoluant différemment suivant des contextes donnés, typiquement des schémas basés sur le paradigme de communication des appels de procédure à distance. En effet, ces schémas non prédictibles ne pouvant être pré-cablés par un programmeur averti au moment même du développement de l'application sont dépendants d'un ordonnancement s'accommodant de la situation courante.

5.5. *Distribution des messages sur des réseaux hétérogènes*

L'architecture de NEWMADELEINE intègre nativement la possibilité d'exploiter simultanément plusieurs réseaux physiques. Cela permet d'associer différentes tech-

Du côté du client,

```
int nb_pages = random(), numero_page;
bool trouve;

for(i = 0; i < nb_pages; i++){
  numero_page = random();

(1) pack(destination,
         mon_id, sizeof(int));
(2) pack(destination,
         numero_page, sizeof(int));
(3) pack(destination,
         envoi_diff, sizeof(bool));
(4) pack(destination,
         type_acces, sizeof(int));

(5) unpack(&numero_page, sizeof(int));
(6) unpack(&trouve, sizeof(bool));
  if(trouve){
(7)   unpack(&page, taille_page);
  } else {
    // recherche de page
    // sur un autre noeud
  }
}
```

Du côté du serveur,

```
int source, numero_page, type_acces;
bool envoi_diff;

while(1){
(1) unpack(&source, sizeof(int));
(2) unpack(&numero_page, sizeof(int));
(3) unpack(&envoi_diff, sizeof(bool));
(4) unpack(&type_acces, sizeof(int));

 page = recherche_page(numero_page,
                       type_acces);

(5) pack(source,
         numero_page, sizeof(int));
  if(page){
(6)   pack(source,
           trouve, sizeof(bool));
(7)   pack(source,
           page, taille_page);
  } else {
(8)   pack(source,
           pas_trouve, sizeof(bool));
  }
}
```

Figure 6. *Code d'une série de demande de pages auprès d'une MVP.*

nologies réseau à différents canaux logiques (parfois appelés « *communicateurs* ») de l'application si ces derniers ont des contraintes particulières de latence ou de débit par exemple. Cela permet également d'utiliser plusieurs technologies réseau *simultanément* afin d'effectuer plusieurs transmissions en parallèle et ainsi augmenter le débit pour un même canal logique : c'est la technique des communications multi-rails.

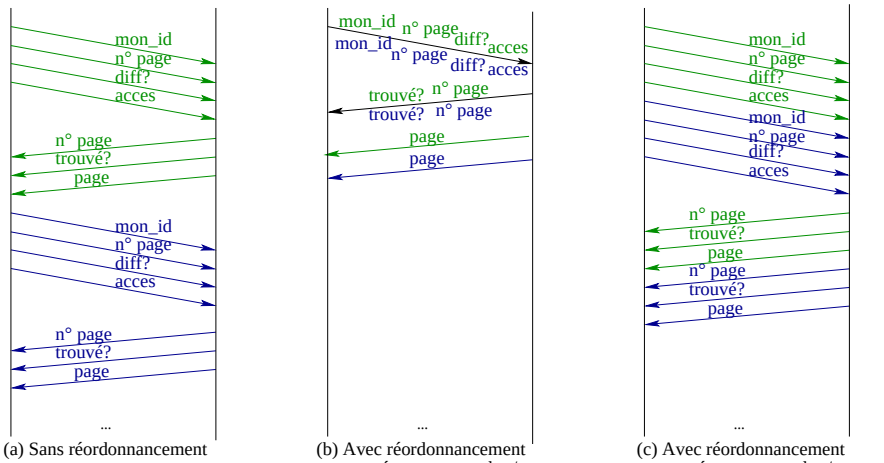


Figure 7. *Schémas de communication de la demande de 2 pages auprès d'une MVP*

	MYRI-10G		QUADRICS	
Ordonnancement	Sans	Avec	Sans	Avec
Temps de transfert	105 μs	69 μs	63 μs	21 μs
Gain	25%		65%	

Figure 8. *Résultats obtenus pour la demande et la réception de 3 pages de 65 Ko.*

L'ordonnanceur de NEWMADELEINE étant piloté par l'activité des cartes réseau, il est très facile de développer une stratégie capable de distribuer les paquets à transmettre de manière *gloutonne* : dès qu'une carte réseau devient libre, on l'alimente tout simplement avec le prochain paquet à envoyer. Lorsque deux cartes sont disponibles et qu'un seul paquet est prêt à être envoyé, ce dernier peut être coupé en deux pour alimenter les deux cartes simultanément.

En suivant ce principe, nous avons réalisé une première implémentation naïve de cette stratégie, et l'avons évaluée au-dessus de notre plate-forme bi-rails équippée des réseaux MYRI-10G et QUADRICS. Les premières évaluations de ce prototype, reportées sur les figures 9 et 10 ont toutefois montré qu'il est nécessaire de raffiner la stratégie à la fois pour les petits et pour les gros messages:

Petits messages La figure 9 montre les performances obtenues lorsque l'on distribue de manière gloutonne deux segments de taille égale sur notre plate-forme multi-rail (courbe « Multi-rail glouton »). En comparaison, nous avons reporté le temps de transfert obtenu lorsque l'on agrège ces deux segments (par copie) et

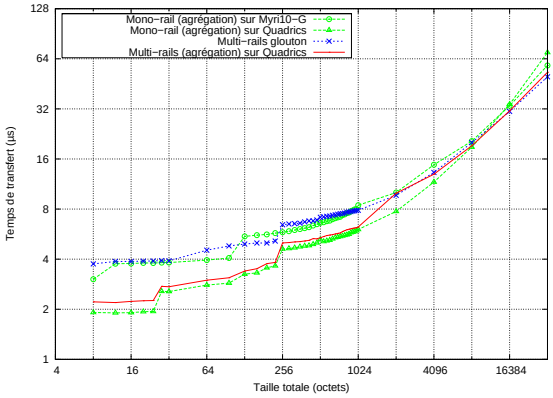


Figure 9. *Envoi des agrégations de petits messages sur le réseau le plus rapide et distribution des longs sur l'ensemble des cartes disponibles - Latence.*

que l'on envoie le segment résultant en mode mono-rail (courbes « Mono-rail »). Les résultats montrent qu'il n'y a aucun intérêt à utiliser le multi-rail pour des segments dont la taille cumulée est inférieure à 32 Ko. La cause tient à la façon dont les messages sont envoyés de la mémoire centrale vers la carte réseau du côté émetteur : pour les petits messages, les bibliothèques de communications telles que MX ou ELAN utilisent le mode PIO (*Programmed Input-Output*) qui monopolisent le processeur et dégradent la bande passante disponible sur le bus mémoire durant l'opération. Ainsi, l'émission des deux segments est quasiment sérialisée.

Gros messages La figure 10 montre les performances obtenues lorsque l'on coupe un gros segment en deux fragments de même taille (courbe « Multi-rail homogène ») pour les envoyer sur deux réseaux différents. En comparaison, nous avons reporté le temps de transfert obtenu en transmettant le segment intact sur un seul des deux réseaux (courbes « Mono rail »). Les résultats montrent que l'on obtient un gain important en terme de débit : pour un message de 8 Mo, le débit obtenu s'élève à 1670 Mo/s en multi-rail, alors qu'il ne dépasse pas 1170 Mo/s sur MYRI-10G et 850 Mo/s sur QUADRICS. Toutefois, ce gain est inférieur à ce que l'on pourrait espérer en utilisant les deux réseaux au maximum de leur capacité.

Nous avons donc modifié notre stratégie afin d'agréger les petits segments ($<$ 32 Ko) pour les transmettre sur le réseau le plus rapide d'une part, et de scinder les gros segments ($>$ 128 Ko) en respectant un ratio correspondant au débit des cartes réseau pour l'intervalle de taille considéré. Ce dernier point garantit que les cartes réseaux « travaillent » pendant la même durée pour envoyer le fragment qui leur a été affecté, et donc que le temps de transmission est minimal.

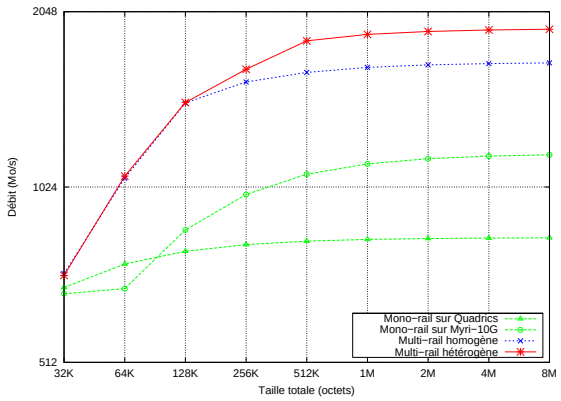


Figure 10. *Messages fractionnés suivant un ratio adapté - Débit.*

La figure 9 montre que notre nouvelle stratégie (courbe « Multi-rail sur Quadrics ») est bien meilleure que la version précédente. Le faible écart avec la courbe « Mono-rail sur Quadrics » s'explique par le simple coût, dans le cas multi-rail, de la scrutation inutile mais systématique de tous les réseaux du côté du récepteur.

Enfin, la figure 10 montre que l'utilisation d'une tactique plus élaborée pour la gestion des gros segments obtient d'excellents résultats (courbe « Multi-rail hétérogène ») : le débit obtenu dépasse 1900 Mo/s, ce qui est très proche de la limite théorique accessible au niveau matériel (< 2 Go/s).

Le point essentiel demeure que la plate-forme NEWMADELEINE facilite grandement ce genre de construction incrémentale de stratégie, sans jamais nécessiter de développer du code dépendant des technologies, tout en garantissant un surcoût logiciel moindre par rapport aux pilotes sous-jacents.

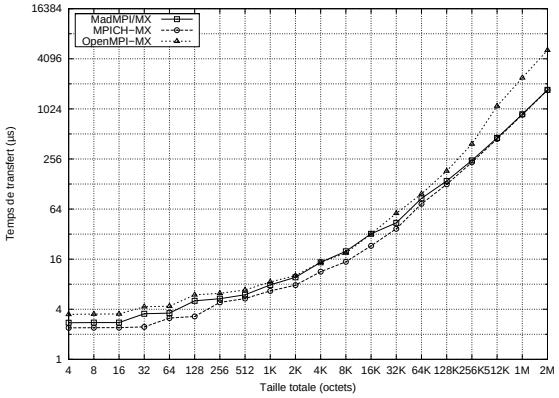
5.5.1. NEWMADELEINE *face à des implémentations de* MPI.

Nous proposons à présent de confronter MAD-MPI aux implémentations les plus courantes de MPI : MPICH et OPEN MPI. Nous allons tout d'abord procéder à un ping-pong pour déterminer le surcoût logiciel de MAD-MPI avant de valider les optimisations mises en place dans le schéma de communication des types indexés (*datatype*).

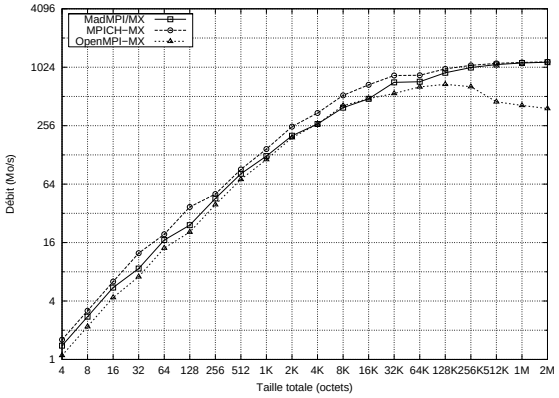
5.5.1.1. Surcoût brut par message élémentaire.

Nous mesurons le surcoût logiciel induit par MAD-MPI à l'aide d'un ping-pong en la comparant à MPICH et OPEN MPI sur MYRI-10G(figure 11) et sur QUADRICS (figure 12).

De même que NEWMADELEINE face à MX et ELAN, MAD-MPI engendre un surcoût logiciel de 0,5 μs par rapport aux deux implémentations de MPI évaluées et



(a) Latence.



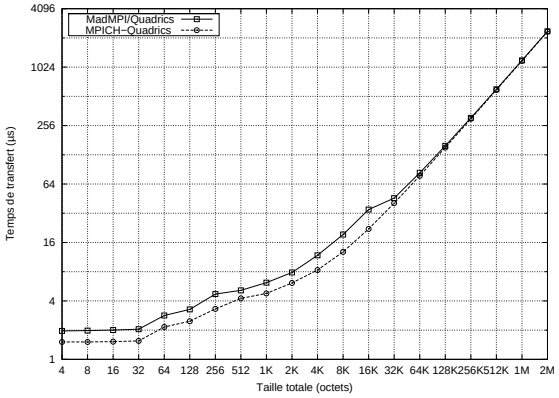
(b) Bande passante.

Figure 11. MAD-MPI *vs* MPICH *vs* OPENMPI *sur* MYRI-10G.

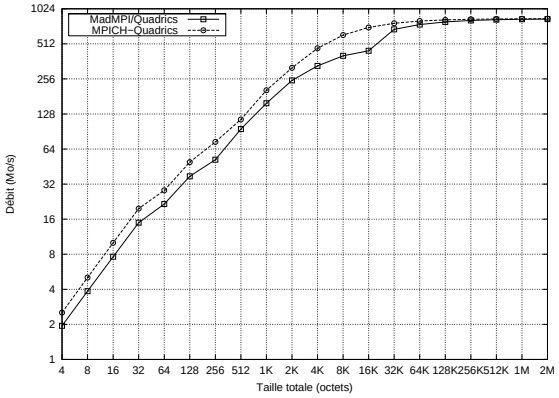
atteint les mêmes niveaux de bande passante : 1170 Mo/s sur MYRI-10G et 837 Mo/s sur QUADRICS.

5.5.1.2. Exploitation des types indexés de MPI.

Les types indexés de MPI permettent de décrire un ensemble de zones mémoire non contigües à envoyer. MPICH copie systématiquement l'ensemble des fragments décrits dans un même tampon avant de l'envoyer en une seule transaction pour ensuite les redistribuer en réception (Gropp *et al.*, n.d.). Ceci est particulièrement efficace tant que le temps de copie nécessaire reste inférieur à celui qui aurait été engendré par l'envoi indépendant de chaque zone. Cependant, le coût d'une copie étant proportionnel à la masse des données considéré, ce mécanisme devient rapidement cher dès que la taille cumulée des fragments augmente. En l'absence de documentation,



(a) Latence.

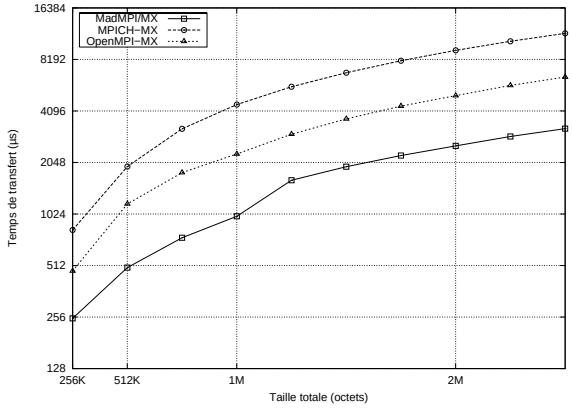


(b) Bande passante.

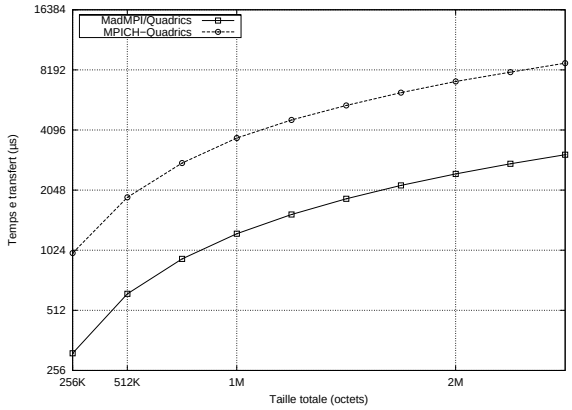
Figure 12. MAD-MPI *vs* MPICH *sur* ELAN/QUADRICS.

nous supposons que OPEN MPI suit le même protocole. De son côté, MAD-MPI utilise un algorithme qui génère une requête de communication pour chaque fragment de donnée (Furmento *et al.*, 2005), ce qui permet à NEWMADELEINE de produire un schéma de communication optimisé à partir des stratégies dont elle dispose. En l'occurrence, nous utilisons la stratégie d'agrégation dans cette expérimentation. Ainsi, tous les blocs de petite taille et les demandes de *rendez-vous* des blocs longs vont être agrégés afin d'être envoyés en une seule communication et redistribués vers leur destination finale à la réception, et les messages longs sont directement reçus dans les zones mémoire indiquées par l'application.

Afin de mettre en valeur les gains de ces optimisations, nous construisons un type indexé qui est une séquence de deux blocs de données : un bloc de 64 octets suivi d'un de 256 Ko. Cette séquence est répétée plusieurs fois afin d'obtenir un nombre de



(a) Type indexé sur MYRI-10G.



(b) Type indexé sur QUADRICS.

Figure 13. *Type indexé.*

blocs conséquent. Nous présentons dans la figure 13 les performances obtenues pour un ping-pong où sont échangés un tel type indexé de données. Un gain de près de 70 % peut être observé par rapport à MPICH-MX, 50 % par rapport à OPENMPI-MX et jusqu'à 70 % par rapport à MPICH-QUADRICS.

6. Travaux apparentés

Nombre de travaux de recherche ont été menés dans le but d'exploiter au mieux les réseaux rapides.

MPICH2-NEMESIS(Buntinas *et al.*, 2006) est à notre connaissance la meilleure implémentation actuelle de MPI en terme de latence et de débit. Cependant, ordonnan-

cer les communications ni même les multiplexer n'est en accord avec leur orientation latence. MPICH n'a pas les mêmes classes d'application cibles.

PM VERSION 2(Takahashi *et al.*, 2000) est une bibliothèque de communication de bas niveau. Elle permet l'exécution d'une même application sur différents réseaux. Cependant, PM2 est concue pour être la base de YAMPII, une implémentation de MPI produite par la même équipe, également orientée latence.

VMI 2(Pakin *et al.*, 2002a) est une bibliothèque de communication bas-niveau mais orientée tolérance aux pannes. Ainsi, elle est capable de faire de l'équilibrage de charge sur des réseaux homogènes et hétérogènes. Elle ne multiplexe pas les flux de communication mais fait de l'ordonnancement entre les cartes réseau, ce qui une politique d'ordonnancement orienté débit.

MADELEINE 3 (Aumage *et al.*, 2002) est la version antérieure de notre propre NEWMADELEINE. Elle propose déjà à travers son interface enrichie des moyens d'agglomérer des messages n'ayant pas d'inter-dépendances. Cependant, le réordonnancement impliquant des inversions de paquets n'est pas envisageable car les données transitent sous leur forme brute. De plus, le multiplexage n'y est pas géré nativement.

De plus, plusieurs projets ont travaillé autour de l'exploitation simultanée de technologies réseaux hétérogènes.

LA-MPI(Aulwes *et al.*, 2004; Coll *et al.*, 2001) est une implémentation de MPI capable d'envoyer des messages sur des réseaux hétérogènes. Elle est également capable de découper un message et de l'envoyer sur différentes ressources de communication mais uniquement de même technologie. Ce projet fait à présent partie du consortium OPEN MPI, qui lui aussi, gère le découpage de messages mais propose l'envoi des fragments sur des technologies hétérogènes.

MPICH-VMI2(Pakin *et al.*, 2002a)(Pakin *et al.*, 2002b) est également une implémentation de MPI qui propose un support de gestion des communications sur réseaux hétérogènes.

Pour finir, MUNICLUSTER (Mohamed, 2005) permet d'équilibrer la charge entre plusieurs cartes réseaux de même technologie ou non et met en prtaique des stratégies de découpages de messages mais seulement au dessus des interfaces sockets.

7. Conclusion

Si l'exploitation brute des réseaux rapides pour des requêtes simples est à présent parfaitement maîtrisée par des bibliothèques de communication dotées d'implémentations hyper-optimisées, de nombreux gains de performances sont encore possibles si l'on considère dans leur *globalité* les flux de communications entre paires de processus.

Dans cet article, nous présentons une nouvelle architecture de communication pour réseaux rapides nommée NEWMADELEINE. Sa principale originalité réside dans son

moteur d'optimisation des transferts de données qui, tout en exploitant les informations fournies par l'interface (dépendances entre segments, contraintes temporelles), permet d'appliquer dynamiquement des stratégies d'optimisation génériques. La bibliothèque est multithreadée, ce qui lui permet, en fonction de l'activité des cartes réseau d'accumuler des requêtes lorsque les cartes sont occupées, puis de déclencher les optimisations *just-in-time* lorsqu'une ou plusieurs cartes deviennent disponibles. De plus, entre chaque paire de processus, les optimisations ont une portée globale aux flux de données, quel que soit le nombre de canaux de communications utilisés, ce qui permet des optimisations opportunistes et aggressives.

Les évaluations préliminaires de l'implémentation sur MX/Myrinet et Elan/Quadrics — et de façon plus impressionnante encore lorsque les deux réseaux sont utilisés conjointement en multi-rail — montrent que les bénéfices de l'approche sont sensibles sur des exemples rencontrés dans des applications réelles. Dans un futur proche, des évaluations plus vastes seront menées au sein de l'environnement PadicoTM (Denis *et al.*, 2003), qui promet d'être un terrain d'optimisations potentielles très vaste en matière de multiplexage des communications.

Ce travail ouvre de nombreuses perspectives. La plus excitante concerne la conception d'un algorithme capable d'évaluer, en temps réel, la meilleure combinaison de stratégies à appliquer à tout moment, pour une fenêtre de messages bornée. Le problème de combinatoire sous-jacent est très difficile dans le cas général, mais nous avons déjà déterminé des algorithmes polynomiaux dans les cas simples (lorsqu'une seule stratégie réalise des agrégations de paquets).

8. Bibliographie

Aulwes R. T., Daniel D. J., Desai N. N., Graham R. L., Risinger L. D., Taylor M. A., Woodall T. S., Sukalski M. W., « Architecture of LA-MPI, A Network-Fault-Tolerant MPI. », *Proc. 18th International Parallel and Distributed Processing Symposium (IPDPS 2004)*, April, 2004.

Aumage O., Bougé L., Denis A., Eyraud L., Méhaut J.-F., Mercier G., Namyst R., Prylli L., « A Portable and Efficient Communication Library for High-Performance Cluster Computing », *Cluster Computing*, vol. 5, n° 1, p. 43-54, 2002.

Buntinas D., Mercier G., Gropp W., « Design and Evaluation of Nemesis: a Scalable, Low-Latency, Message-Passing Communication Subsystem », *Proc. 6th IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2006)*, Singapour, May, 2006.

Coll S., Frachtenberg E., Petrini F., Hoisie A., Gurvits L., « Using Multirail Networks in High-Performance Clusters », *Proceedings of the 3rd IEEE International Conference on Cluster Computing*, IEEE Computer Society, p. 15-24, 2001.

Denis A., Pérez C., Priol T., « PadicoTM: An Open Integration Framework for Communication Middleware and Runtimes », *Future Generation Computer Systems*, vol. 19, n° 4, p. 575-585, 2003.

Furmento N., Mercier G., Optimisation Mechanisms for MPICH-Madeleine, Technical Report n° 0306, INRIA, July, 2005. Also available as LaBRI Report 1362-05.

Graham R. L., Woodall T. S., Squyres J. M., « Open MPI: A Flexible High Performance MPI », *Proceedings, 6th Annual International Conference on Parallel Processing and Applied Mathematics*, 2005.

Gropp W., Lusk E., Swider D., Toward Faster Packing and Unpacking of MPI Datatypes, Technical report, n.d. http://www.mcs.anl.gov/mpi/mpich1/papers/index.html.

Mohamed N., « Self-Configuring Communication Middleware Model for Multiple Network Interfaces », *29th Annual International Computer Software and Applications Conference (COMPSAC'05)*, 2005.

Myricom, Inc., Myrinet EXpress (MX): A High Performance, Low-level, Message-Passing Interface for Myrinet, Technical report, 2003.

Pakin S., Pant A., « VMI 2.0: A Dynamically Reconfigurable Messaging Layer for Availability, Usability, and Management », *8th International Symposium on High Performance Computer Architecture (HPCA-8)*, 2002a.

Pakin S., Pant A., « VMI 2.0: A Dynamically Reconfigurable Messaging Layer for Availability, Usability, and Management », *The 8th International Symposium on High Performance Computer Architecture (HPCA-8), Workshop on Novel Uses of System Area Networks (SAN-1)*, Cambridge, Massachusetts, February 2,, 2002b.

Quadrics Ltd., Elan Programming Manual, Technical report, 2003.

Takahashi T., Sumimoto S., Hori A., Harada H., Ishikawa Y., « PM2: High Performance Communication Middleware for Heterogeneous Network Environments », *SC'00*, p. 52-53, 2000.