CORBA haute performance

Alexandre Denis

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

CORBA haute performance

Alexandre Denis

N˚4553

Septembre 2002

THÈME 1

ISSN 0249-6399 ISRN INRIA/RR--4553--FR

CORBA haute performance

Alexandre Denis*

Thème 1 — Réseaux et systèmes
Projet PARIS

Rapport de recherche n ° 4553 — Septembre 2002 — 32 pages

Résumé : Les codes de calcul étant de plus en plus complexes, il devient courant de modulariser le code en couplant des modules à l'aide d'un middleware, par exemple CORBA. Les implémentations existantes de CORBA exploitent "efficacement" les réseaux locaux et longue distance grâce à TCP/IP. Cependant, quand il s'agit d'exécuter différents codes couplés sur une grappe de PC, il serait intéressant que CORBA puisse exploiter directement les réseaux haut débit tels que SCI ou Myrinet. Cet article présente une implémentation CORBA reposant sur omniORB en utilisant la bibliothèque de communications Madeleine. Nous avons mis l'accent sur les performances et la transparence d'utilisation. Ainsi, nous intégrons à l'ORB un support de protocoles multiples avec sélection automatique, et nous maintenons l'interopérabilité avec les autres ORB. Les performances sont au rendez-vous. Sur Ethernet-100, le débit plafonne à 11.6 Mo/s (92 Mbit/s) pour une latence à 160 microsecondes ; notre implémentation réalise des transferts à 240 Mo/s (1.9 Gbit/s) sur Myrinet-2000, avec une latence abaissée à 20 microsecondes.

Mots-clé : CORBA, réseaux haut débit, grappes de PC, couplage de codes, objets distribués.

(Abstract: pto)

Publié dans : RSTI série TSI, Volume 21, no 5/2002, pages 659-683, éditions Hermès Science.

* Alexandre.Denis@irisa.fr

Unité de recherche INRIA Rennes
IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex (France)
Téléphone : 02 99 84 71 00 - International : +33 2 99 84 71 00
Télécopie : 02 99 84 71 71 - International : +33 2 99 84 71 71

High Performance CORBA

Abstract: Numerical simulation codes become more and more complex. It is now wide-spread to couple several codes; code coupling is performed by a middleware, for example CORBA. Existing CORBA implementations can efficiently utilize SAN and WAN thanks to TCP/IP. However, for code coupling on a cluster of PC, it would be convenient that the ORB utilize high speed networks such as Myrinet and SCI. This paper presents an implementation based on omniORB and Madeleine. Transparency is our main concern: protocol auto-selection and interoperability with other ORB. The performance is excellent: the bandwidth goes as high as 1.9 Gbit/s on Myrinet-2000, with a latency lowered to 20 microseconds.

Key-words: CORBA, high performance networks, clusters of workstations, code coupling, distributed objects.

1 Couplage de code par CORBA sur une grille de calcul

Le développement du Web a révolutionné la façon dont nous pensons l'accès à l'information. Il est aujourd'hui possible d'accéder à virtuellement n'importe quelle information électronique dans le monde via le Web. Les grilles de calcul proposent une révolution similaire pour les moyens de calcul. Un des ouvrages fondateurs du *Grid computing* [17] définit une grille comme un ensemble de supercalculateurs et de grappes de PC dispersés partout dans le monde, reliés par Internet, et disposant de grandes capacités de stockage. Une grille de calcul est composée de ressources distribuées sur des sites différents. En raison de l'hétérogénéité des ressources, la conception d'un environnement exécutif pour cette infrastructure est une tâche difficile. Dans cet article, nous nous intéressons particulièrement à une facette qu'un environnement exécutif pour la grille doit prendre en compte : permettre l'exécution efficace des codes.

Les ressources de communication d'une grille de calcul couvrent un large spectre de technologies, allant des WAN (*Wide Area Network*) tels que le réseau expérimental français VTHD à 2.5 Gbit/s, jusqu'aux LAN (*Local Area Network*) ou SAN (*System Area Network*) tels que Myrinet [9] ou SCI [18]. Les technologies sont différentes, les performances sont différentes, les protocoles sont différents. Pourtant, il serait souhaitable que l'utilisateur n'aie pas à gérer lui-même ces différences. Nous pensons qu'il est plus raisonnable que le modèle de programmation prenne en compte cette diversité : pour faciliter le déploiement sur des ressources hiérarchiques, l'application est écrite sous forme hiérarchique.

Les applications pour la grille peuvent demander des modèles de programmation variés, et donc des exécutifs différents pour que les programmeurs aient à leur disposition le modèle de programmation adapté à leurs besoins. Même si les premières implémentations d'infrastructures pour la grille, telles que Globus [16], reposaient essentiellement sur le paradigme de passage de messages, nous pensons que les applications futures demanderont d'autres paradigmes tels que les objets distribués ou les composants logiciels. Parmi les applications destinées à une exécution sur une grille de calcul, les simulations multiphysiques sont un bon exemple. Elles sont composées de plusieurs codes de simulation couplés ; par exemple, chaque code simule un paramètre physique. Considérons la conception d'un satellite. Elle nécessite les simulations de la mécanique, de la thermique, de la dynamique et de l'optique. Chaque simulation est réalisée par un code distinct, éventuellement parallèle. Ce type d'application semble bien adapté à une grille de calcul puisque chaque code peut s'exécuter sur une machine parallèle ou vectorielle. Les codes parallèles utilisent MPI, PVM, ou une DSM. Le couplage des différents codes peut être réalisé par des mécanismes

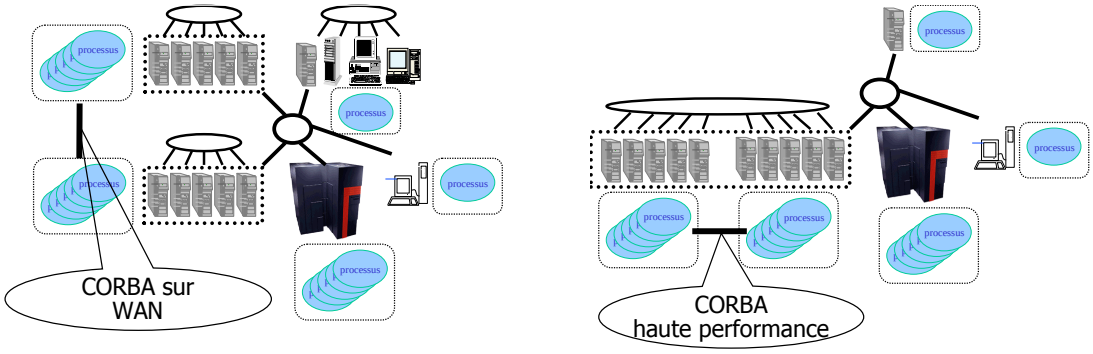

FIG. 1 – CORBA *doit s'adapter au réseau selon le déploiement*

d'invocation de méthode à distance (tels que RMI Java ou CORBA). Dans cet article, nous proposons d'utiliser CORBA pour réaliser le couplage de code. CORBA présente des caractéristiques intéressantes : il prend en charge l'hétérogénéité des machines, il utilise une approche orientée objet, et il offre une interopérabilité entre plusieurs langages. Il permet également un couplage de codes parallèles [11].

Quand les codes parallèles sont déployés, selon la disponibilité des machines, les liens de couplage empruntent un réseau longue distance (WAN) ou un lien rapide d'une grappe (SAN, tel que Myrinet [9] ou SCI [18]). Pour qu'un modèle de programmation par codes parallèles couplés par CORBA soit performant à l'exécution, il est nécessaire de disposer d'une implémentation CORBA haute performance capable de s'adapter au réseau disponible comme le montre la figure 1. La structuration de l'application ne pénalisera pas les performances si CORBA obtient des performances similaires aux performances des communications MPI.

Les implémentations CORBA actuelles sont conçues dans le but d'une utilisation sur réseaux longue distance et réseaux locaux. Elles ne prennent pas en charge les réseaux haute performance et ne sont donc pas capables d'exploiter toutes les ressources disponibles sur les grappes et les machines parallèles. Nous proposons une implémentation CORBA haute performance capable de tirer profit de ces réseaux. Quelques travaux ont déjà été réalisés dans le domaine de CORBA haute performance. Par exemple, OmniORB [21, 25] et TAO [20] ont été adaptés au réseau ATM et CrispOrb [19] est destiné au réseau Fujitsu Synfinity-0. Cependant, ces travaux n'ont pas abouti à des résultats concluants. De plus, ils se restreignent chacun à un type de réseau particulier. Nous proposons une nouvelle approche qui combine la

portabilité sur différents types de réseaux et la performance grâce à l'utilisation de la bibliothèque Madeleine [7] ; nous maintenons l'interopérabilité avec les implémentations CORBA classiques ; enfin, à l'aide de Padico [12], nous gérons la coopération entre les exécutifs CORBA et MPI pour mettre en œuvre un modèle de programmation hiérarchique qui combine couplage par CORBA et parallélisme par MPI.

Après avoir présenté les notions fondamentales de CORBA dans la section 2, nous analysons les performances des implémentations CORBA en section 3 pour guider nos choix lors de l'implémentation haute performance. La section 4 présente une implémentation de CORBA haute performance que nous avons réalisée. Nous présentons les travaux connexes en section 5. Enfin, nous concluons en section 6.

2 Présentation de CORBA

2.1 Principes de CORBA

CORBA [23] est l'acronyme de *Common Object Request Broker Architecture* – architecture commune pour courtier de requêtes objets. CORBA est une norme définie par l'OMG [4] (*Object Management Group*), consortium ouvert qui réunit des industriels et des universitaires et qui a pour but de normaliser les technologies orientées objet. Il existe de nombreuses implémentations de la norme CORBA (voir la section 2.2), dont certaines sont des logiciels libres. La dernière version est CORBA 2.5. CORBA étant une norme, l'OMG fait passer aux différentes implémentations un test de conformité à la norme. Une implémentation qui passe le test avec succès est dite "certifiée".

CORBA définit une architecture d'objets distribués. Cette architecture est indépendante du type de machine, du système d'exploitation, et du langage de programmation utilisé. L'architecture CORBA est composée essentiellement de l'ORB (*Object Request Broker* – courtier de requêtes objet) qui est le cœur de l'architecture CORBA, et des objets pris en charge par l'ORB. L'ORB assure le transport des requêtes sur le réseau. Du point de vue du programmeur, n'importe quel objet CORBA est accédé de façon transparente, comme si c'était un objet local. L'ORB se charge d'intercepter les invocations de méthode, de localiser l'objet, d'acheminer par le réseau les invocations avec leurs paramètres et de transmettre les éventuelles valeurs de retour des méthodes. Les objets définis par le programmeur sont pris en charge par l'ORB. L'interface de chaque objet est spécifiée à l'aide du langage IDL (*Interface Definition Language*) qui permet l'indépendance par rapport au langage d'implémentation.

Pour que différentes implémentations CORBA puissent communiquer entre elles, la norme spécifie un protocole de transport des requêtes. Ce protocole s'appelle GIOP

(*General Inter*-ORB *Protocol*). Il spécifie un encodage des paramètres d'entrée et des valeurs de retour des méthodes, ainsi qu'un format de messages. L'incarnation de GIOP sur TCP/IP s'appelle IIOP (*Internet Inter*-ORB *Protocol*). S'il le juge utile, un ORB peut utiliser n'importe quel protocole pour les communications entre un client et un serveur. S'il s'affranchit de GIOP, on parle alors d'ESIOP (*Environment-Specific Inter*-ORB *Protocol*). Pour l'interopérabilité, il est cependant nécessaire que chaque objet puisse être aussi accédé par IIOP. Par souci de simplicité, la plupart des ORB utilisent exclusivement IIOP (et donc TCP/IP) pour toutes leurs communications.

2.2 Implémentations CORBA sur TCP/IP

Nous présentons ici quatre implémentations largement répandues et qui sont disponibles gratuitement. Nous n'avons utilisé que des implémentations en langage C++.

- MICO [1], développé à l'université de Francfort (Allemagne), présente l'avantage d'être une implémentation complète OpenSource, respectant strictement la norme CORBA 2.3. Il n'est pas réputé être le plus performant, mais ses performances sont toutefois honorables.
- TAO [29] (The ACE ORB) est développé à l'université de Washington (USA). ACE est un environnement qui prend en charge la plupart des problèmes de portabilité et fournit une abstraction des ressources (réseaux, threads, etc.) à l'application. TAO est OpenSource, et est un ORB complet certifié CORBA 2.3 destiné principalement aux applications "temps réel".
- Distribué par Iona, ORBacus [2] est un ORB certifié CORBA 2.3. Il est gratuit pour une utilisation non commerciale. Il s'appelait auparavant OmniBroker.
- OmniORB [5] est développé aux laboratoires AT&T de Cambridge (Royaume-Uni). C'est un ORB OpenSource. Il est particulièrement simple et efficace, mais néanmoins certifié conforme aux spécifications CORBA 2.1. Nous nous basons sur la version 3. La version 4 qui implémentera CORBA 2.4 est annoncée prochainement.

3 Analyse des performances de CORBA

Pour guider nos choix vers une implémentation CORBA haute performance, nous commençons par analyser les ORB existants. Ceci nous permet d'identifier les parties qui ont le plus d'influence sur la performance. Pour cela, nous nous basons sur les quatre ORB présentés dans la section précédente.

3.1 Caractérisation des performances

L'OMG suggère une méthodologie [22] de mesure des performances des implémentations CORBA, et de l'ORB en particulier. Nous proposons un benchmark qui respecte les principes de base donnés par l'OMG, à savoir : adéquation avec l'utilisation typique, portabilité, scalabilité et simplicité. L'utilisation que nous visons pour CORBA est le calcul haute performance. Nous ne porterons donc pas notre attention aux performances des divers services de CORBA qui ne sont pas utilisés dans ce cadre. Nous nous concentrerons plutôt sur le cœur de l'ORB : l'invocation d'une méthode distante. Notre objectif étant de montrer qu'utiliser CORBA pour structurer les applications ne dégrade pas les performances, nous serons amenés à comparer CORBA et MPI. Il est donc naturel d'utiliser les mêmes métriques que celles utilisées pour évaluer MPI plutôt que la métrique habituelle dans le monde CORBA (nombre de requêtes par seconde). Nous retenons comme grandeurs caractéristiques la latence et le débit que nous définissons ci-dessous.

Notons T le temps d'invocation d'une méthode à distance. Il se décompose en plusieurs phases :

- latence d'invocation (t_1) qui correspond au temps d'émission sur le réseau de l'entête qui annonce quelle méthode et quel objet distant utiliser ;
- temps d'envoi t_{in} des paramètres *in* (paramètres d'entrée) de la méthode, que l'on peut caractériser par son débit D_{in} ;
- temps d'exécution t_{exec} de la méthode ;
- temps d'envoi t_{out} en retour des paramètres *out* (paramètres de sortie), que l'on peut caractériser par le débit D_{out} ;
- latence de notification de terminaison de la méthode distante (t_2).

À partir de cette décomposition du temps d'invocation d'une méthode distante, définissons la latence et le débit dans le monde CORBA :

- Latence et RTT (RTT=*RoundTrip Time*, temps d'aller-retour) : nos mesures consistent en l'invocation distante d'une méthode vide ("ping"). Soit RTT le temps total de cette invocation. Nous avons donc $RTT = t_1 + t_2$. La latence est l'intervalle de temps entre l'invocation dans le client et l'exécution réelle de la méthode dans le serveur. Strictement parlant, la latence est t_1. Cependant, pour faciliter les mesures, nous utilisons une approximation. Nous posons la définition suivante : latence $= \frac{RTT}{2}$;
- Débit : c'est le débit d'envoi des paramètres. Pour le mesurer, nous invoquons une méthode qui accepte en paramètre bidirectionnel ("inout") une taille variable de données de taille S. Le temps total d'invocation de la méthode distante

ORB	MICO 2.3.5	TAO 1.1	ORBacus 4.0.5	OmniORB 3.0.2
Latence (μs)	288	255	253	168
Débit maximal (Mo/s)	9.2	9.3	9.5	11.6

TAB. 1 – *Récapitulatif des performances des* ORB *sur Ethernet-100*

est noté T. Pour chaque ORB (MICO, TAO, ORBacus et OmniORB), nous mesurons $T = f(S)$. Nous avons constaté que les débits *in* et *out* sont identiques. Nous noterons désormais $D = D_{\text{in}} = D_{\text{out}}$. Nous avons alors $T = RTT + 2\frac{S}{D}$ d'où $D = \frac{T-RTT}{2S}$.

3.2 Mesures des performances

Nous comparons ORB par ORB, c'est-à-dire que nous mesurons les communications entre un serveur MICO et un client MICO, un serveur TAO et un client TAO, etc. Pour que la comparaison avec MPI soit équitable, nous nous limitons au cas des grappes homogènes dotées de réseaux locaux et/ou de réseaux haute performance. Le cas des architectures hétérogènes et/ou des réseaux longue distance est un tout autre problème que nous ne traitons pas ici. De plus, nous ne nous intéressons pas ici aux performances des autres fonctionnalités de CORBA que nous n'utilisons pas dans le cadre du calcul scientifique.

Notre plate-forme de test est composée de machines bi-processeur Pentium II 450 MHz équipées de cartes Ethernet-100. La figure 2 présente les débits obtenus par les quatre ORB pour le transfert d'une séquence d'entier `Long` 32 bits.

On observe d'une part MICO, TAO et ORBacus, et d'autre part OmniORB nettement plus rapide. En particulier, le débit maximal d'OmniORB est à 11.6 Mo/s alors que les autres plafonnent à environ 9.5 Mo/s. Dans la suite, nous expliquerons ces différences ainsi que l'influence du type des données transmises. Le tableau 1 récapitule les performances de chaque ORB. Nous observons que les hiérarchies de performances pour le débit et pour la latence sont similaires. Sur un réseau local, OmniORB semble clairement être le plus rapide de ces ORB selon les critères de performance qui nous semblent adaptés à notre utilisation.

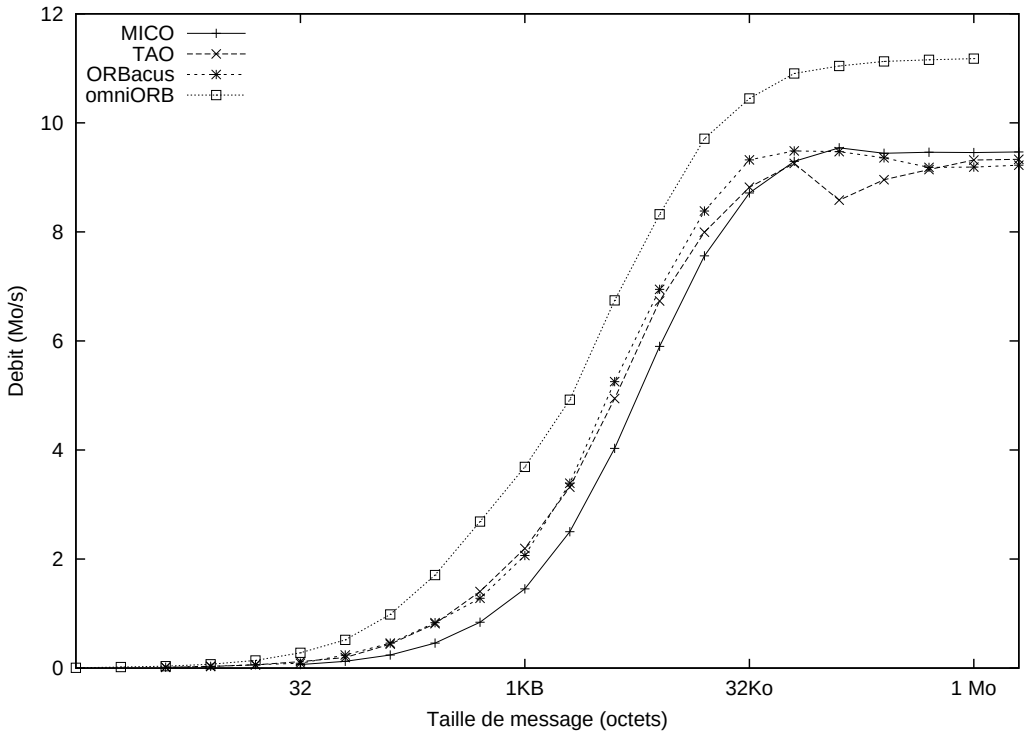

FIG. 2 – *Débits de quatre* ORB *sur Ethernet-100*

3.3 Analyse du débit de transmission

3.3.1 Principe d'analyse

Dans le cadre du couplage de codes de calcul numérique, les données manipulées sont généralement de grande taille. Nous portons donc plus d'intérêt au débit qu'à la latence.

Le débit d'envoi des paramètres est essentiellement conditionné par l'envoi sur le réseau proprement dit, l'encodage (*marshaling*) en émission et le décodage (*demarshaling*) en réception pour assembler et désassembler les requêtes au format GIOP – la plupart du temps, ces opérations réalisent une copie en mémoire à l'émission et une copie à la réception.

Notons D_{encode} le débit d'encodage, $D_{\text{décode}}$ le débit de décodage et D_{net} le débit du réseau. Si on suppose que le débit total ne dépend que de ces trois paramètres, et qu'il n'y a aucun recouvrement entre ces différentes opérations, nous pouvons établir l'égalité suivante :

$$D = \frac{1}{\frac{1}{D_{\text{encode}}} + \frac{1}{D_{\text{net}}} + \frac{1}{D_{\text{décode}}}} \quad (1)$$

D'après les résultats de la section précédente, MICO, TAO et ORBacus ont des performances similaires ; OmniORB est nettement plus rapide. En faisant des mesures supplémentaires (autres types de paramètres), on constate que MICO, TAO et ORBacus ont des comportements semblables. TAO étant le plus gros et le plus complexe des ORB étudiés ici ce qui complexifie l'instrumentation du code, et ORBacus ayant une licence plus restrictive que les autres, nous choisissons de porter notre attention sur MICO qui est représentatif de la stratégie utilisée également par TAO et ORBacus. Nous nous intéresserons ensuite à OmniORB.

3.3.2 Analyse du débit de MICO

L'encodage étant fonction du type, les performances sont variables d'un type à l'autre. Le débit de MICO est donné à la figure 3, en fonction du type des données transmises. Certains types sont simplement recopiés par bloc (les `Long` par exemple) alors que d'autres sont copiés élément par élément (les `Float` par exemple), ce qui explique les grandes disparités entre les différents types. Dans le cas de structures, le débit peut encore diminuer à cause de la recopie élément par élément et de l'appel de méthode virtuelle pour encoder chaque élément.

La figure 4 montre les débits mesurés pour les opérations d'encodage et de décodage de séquences de `Long` (le plus rapide). Des mesures détaillées des copies en

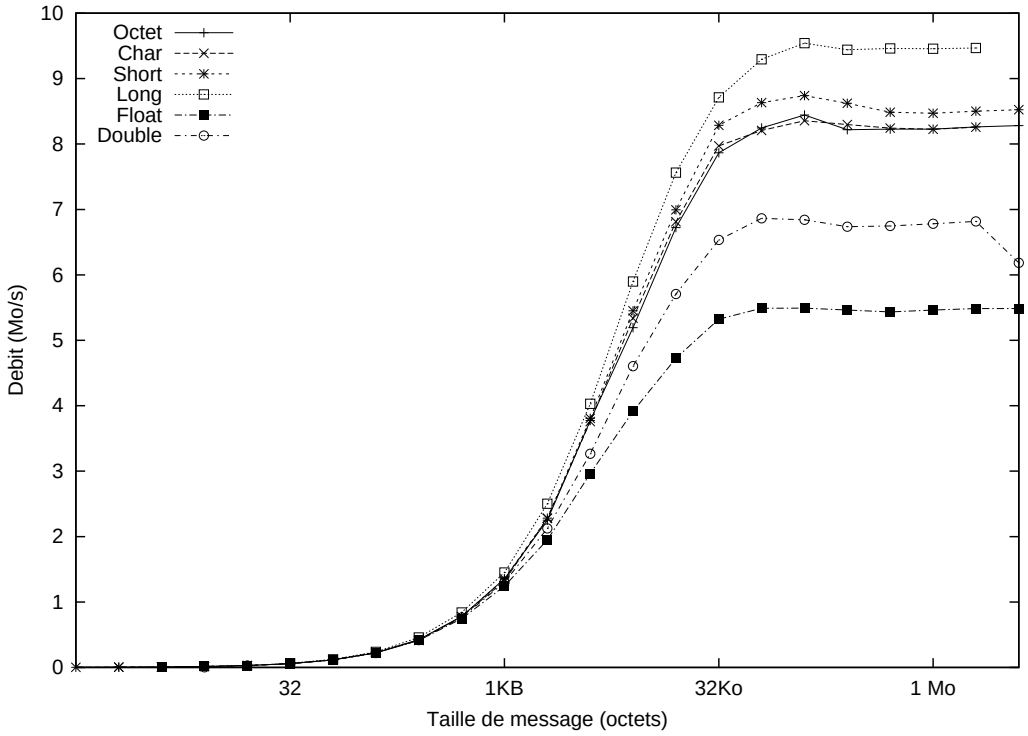

FIG. 3 – *Débit de transmission des paramètres sous* MICO*/Ethernet-100*

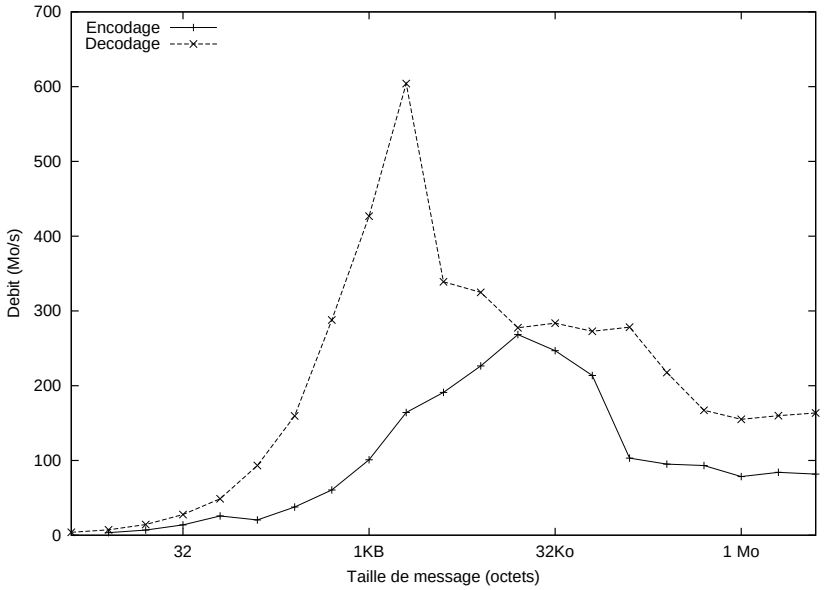

FIG. 4 – *Débits d'encodage et de décodage sous* MICO

mémoire ont montré qu'une partie de ces différences entre D_{encode} et $D_{\text{décode}}$ est due à la présence ou à l'absence de défauts de page, selon que la zone de données est fraîchement allouée ou contient déjà des données. Quand D_{net} est de l'ordre de 10 Mo/s (débit constaté sur Ethernet-100), D_{encode} et $D_{\text{décode}}$ ont une influence négligeable sur le débit total D d'après l'égalité établie précédemment. Ceci est confirmé par l'expérience.

3.3.3 Analyse du début d'OmniORB

OmniORB dépasse le débit donné par l'égalité 1. En réalité, cette égalité n'est pas valable pour OmniORB car dans cet ORB, la vitesse d'encodage ne peut pas être modélisée par un débit. OmniORB utilise une approche "zéro copie" à chaque fois que cela est rendu possible par l'alignement des données en mémoire en particulier. En architecture homogène (ce qui est le cas sur les grappes en général), la stratégie zéro copie est toujours utilisée pour des données suffisamment grandes. Dans ce cas, l'encodage est réalisé en temps constant, D_{encode} et $D_{\text{décode}}$ n'ont aucune signification et le temps d'encodage intervient uniquement sur la latence. Pour OmniORB, on a

donc $RTT = t_{\text{encode}} + t_1 + t_2 + t_{\text{décode}}$, avec t_{encode} et $t_{\text{décode}}$ constants grâce à l'approche zéro copie. On en déduit que pour OmniORB, $D \simeq D_{\text{net}}$ ce qui est vérifié par l'expérience. Même quand le débit du réseau est faible comparé au débit de la mémoire, l'approche sans copie permet un débit meilleur que la stratégie avec une copie.

Pour cette raison, les performances d'encodage d'OmniORB ne dépendent pas du type des données. Sur des architectures homogènes, les séquences d'entiers, de flottants ou même de structures sont envoyées sans copie. On constate expérimentalement que les différents types de données obtiennent exactement les mêmes performances.

Nous avons donc deux modèles différents, l'un adapté à MICO, l'autre adapté à OmniORB. Dans la section suivante, nous confrontons ces deux modèles avec les réseaux haute performance pour justifier le choix d'un ORB plutôt qu'un autre pour une adaptation au haut débit.

4 Implémentation CORBA haute performance

Contrairement aux travaux existants décrits à la section 5, notre but est de réaliser une implémentation CORBA capable d'exploiter efficacement plusieurs types de réseaux haut débit. L'implémentation ne doit pas être spécifique à un type particulier. Nous confions cette tâche à Madeleine, une bibliothèque qui assure la portabilité vis-à-vis du réseau tout en assurant une bonne performance.

4.1 Madeleine

Madeleine [7] est une bibliothèque de communications dédiée aux réseaux hautes performances présents sur les grappes. Elle fait partie de l'environnement de programmation parallèle PM^2 [3] qui inclut également la bibliothèque de multi-threading Marcel. Dans sa deuxième version, Madeleine est capable d'exploiter simultanément plusieurs protocoles parmi SISCI (pour SCI [18]), BIP [26] (pour Myrinet [9]), TCP, VIA [15], SBP et MPI. Des travaux en cours sur Madeleine ont pour but de prendre en charge les grappes de grappes ; la troisième version [8] de Madeleine intègre ces mécanismes multi-grappes.

Les principes fondamentaux de Madeleine sont adaptés à une utilisation sur une grappe. L'accent est mis sur la performance ; les copies de données sont évitées autant que possible pour atteindre un débit utilisable très proche du maximum autorisé par

le matériel. Ceci permet à Madeleine d'atteindre 86 Mo/s sur SCI, 101 Mo/s sur Myrinet et 240 Mo/s sur Myrinet-2000.

Les communications sont organisées autour de canaux de communication. Un *canal* au sens Madeleine est une structure de communication dans laquelle tous les nœuds sont reliés à tous les autres, à la façon d'un *communicateur* MPI. La topologie est statique : il est impossible de rajouter ou de retirer un nœud en cours d'exécution. Ce choix est pertinent dans le cas d'une grappe, car la configuration change rarement en cours de session. La simplicité de mise en place est alors un atout. Par conception, Madeleine impose une configuration homogène pour tous les nœuds, aussi bien du point de vue de l'architecture du processeur que pour les types de réseaux disponibles. Il n'y a pas de conversion de données *little endian – big endian* par exemple.

L'utilisation de Madeleine comme bibliothèque d'accès au réseau nous apporte une interface unique pour un accès efficace aux différents types de réseaux haute performance. Un ORB porté sur Madeleine est d'emblée opérationnel et efficace sur tous les types de réseaux supportés par Madeleine.

4.2 Prototypes haut débit

Avant de réaliser une implémentation complète d'un ORB haute performance, nous réalisons deux prototypes pour évaluer les performances que nous serions en droit d'attendre d'un ORB sur des réseaux haut débit et pour valider nos choix. Dans un premier temps, nous avons utilisé MICO 2.3.3 pour sa simplicité et sa conception modulaire et nous avons analysé ses performances. Nous avons ensuite réalisé un autre prototype basé sur OmniORB 3. Pour virtualiser l'interface des différents réseaux (Myrinet, SCI, VIA, etc.), nous avons choisi la bibliothèque de communications Madeleine décrite ci-dessus. Nous détournons simplement les mécanismes de transfert TCP de l'ORB (appels système `read` et `write`) vers leur équivalent Madeleine. Cette approche est trop simpliste pour une utilisation d'applications CORBA complètes. On ne dispose dans ces premiers prototypes que de deux nœuds, d'aucune interopérabilité, et d'une démarrage de session très rudimentaire. Ces prototypes permettent néanmoins de mesurer de façon réaliste les performances qu'obtiendrait une implémentation complète de CORBA sur Madeleine et sont destinés à guider et valider nos choix.

4.2.1 Prototype basé sur MICO

Le prototype de MICO sur Madeleine a été testée sur les réseaux SCI et Myrinet. La figure 5 présente les débits mesurés en test "ping-pong" pour une séquence d'entiers

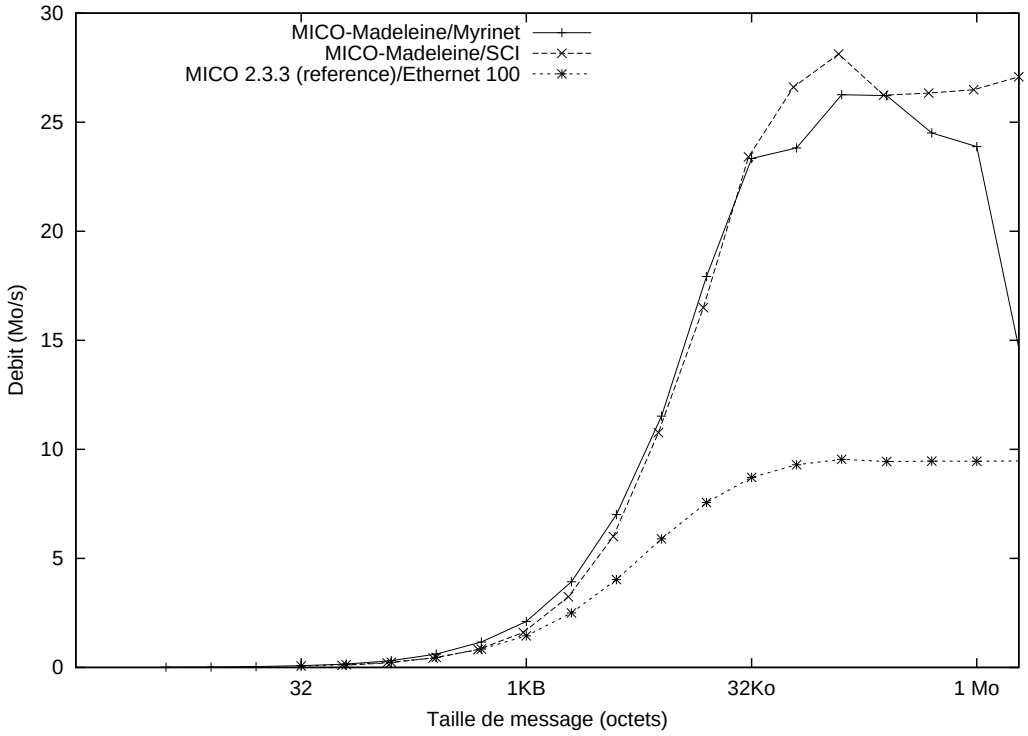

FIG. 5 – *Débit du prototype* MICO*/Madeleine sur Ethernet-100,* SCI *et Myrinet*

32 bits `Long` comme décrit à la section 3.1. Malgré l'utilisation du type `Long` (le plus rapide), ces débits sont décevants quand on les compare au débit brut des réseaux. Le RTT est mesuré à 290 μs (latence 145 μs) sur SCI et 270 μs (latence 135 μs) sur Myrinet, à comparer aux 576 μs (latence 288 μs) sur Ethernet-100. L'amélioration de la latence est appréciable, mais là encore relativement décevante quand on sait que la latence de Madeleine sur Myrinet et SCI est inférieure à 10 μs. On constate par instrumentation du code que l'essentiel de la latence de ce prototype est logiciel quand on utilise des réseaux rapides. Le coût logiciel représente 110 μs du test de latence ; la latence reste médiocre même avec un réseau tel que Myrinet qui descend le coût total du réseau à environ 25 μs lors du test de latence CORBA.

Pour expliquer les débits, le tableau 2 présente différents éléments qui influent sur le débit total. Nous prenons pour D_{encode} et $D_{\text{décode}}$ les débits mémoire mesurés pour de grandes tailles de données, là où l'effet du cache ne se fait plus sentir, respectivement pour une copie en zone statique et en zone fraîchement allouée. Ces valeurs sont confirmées par une instrumentation du code de MICO et correspondent aux mesures de la figure 4. Les variations de D_{encode} et $D_{\text{décode}}$ s'expliquent par le fait que les machines équipées de cartes SCI et Myrinet, bien que dotées de processeurs identiques, sont sensiblement différentes au niveau de la gestion du bus mémoire. Nous prenons pour D_{net} le débit maximum mesuré avec Madeleine. Le débit théorique est celui donné par l'équation 1 calculé à partir des trois valeurs précédentes. Enfin, le débit mesuré est le débit apparent lors du test ping-pong.

Nous constatons sur le tableau 2 que le débit mesuré est très proche du débit maximum calculé théoriquement, ce qui valide notre modèle. Nous exploitons 90 % de la bande passante théoriquement possible. Or, ce modèle montre que les copies sont très coûteuses et limitent la bande passante utilisable à environ 30 % du débit nominal du réseau. Si on applique ce modèle aux machines les plus récentes (copies mémoire mesurées à 260 Mo/s, réseau Myrinet-2000 de débit nominal 250 Mo/s), on retrouve la même borne de l'ordre de 30 % de débit utilisable. Pour dépasser cette

réseau	D_{encode}	$D_{\text{décode}}$	D_{net}	débit théorique	débit mesuré
Ethernet-100	129	80	12	9.6	9.4
SCI	129	80	80	31.4	27.7
Myrinet	113	72	101	30.6	26.2

TAB. 2 – *Analyse du débit en crête de* MICO *en Mo/s*

limite, il est nécessaire de s'affranchir des copies, et de s'orienter vers une approche "zéro copie".

4.2.2 Prototype basé sur OmniORB

Nous avons réalisé un prototype basé sur OmniORB, en utilisant la même méthode que pour MICO : détourner simplement les appels `read` et `write`. Nous obtenons des performances excellentes qui sont inchangées dans la version complète et sont décrites en détail à la section 4.9. Le débit en crête est de 86 Mo/s sur SCI, 101 Mo/s sur Myrinet et 240 Mo/s sur Myrinet-2000. La latence passe de 160 μs sur TCP/Ethernet-100 à 55 μs sur SCI et Myrinet, et 20 μs sur Myrinet-2000.

Ces bonnes performances nous encouragent à réaliser une implémentation complète basée sur OmniORB. En effet, modifier un ORB existant pour lui faire adopter des méthodes zéro copie est une tâche ardue qui impliquerait de modifier des centaines de lignes de code. Il est préférable d'adapter aux réseaux haut débit un ORB qui dispose déjà de ces mécanismes et qui fournit à la fois un haut débit et une faible latence. Ceci explique notre choix d'OmniORB.

4.3 Principe de l'implémentation complète

D'après les résultats de la section précédente, OmniORB est le meilleur candidat pour une implémentation complète CORBA haute performance. Son principal atout est sa stratégie d'encodage/décodage qui ne réalise pas de copie la plupart du temps sur une grappe homogène. Nous basons donc notre implémentation complète de CORBA sur réseaux haut débit sur OmniORB.

Le premier prototype d'OmniORB sur Madeleine, présenté en section 4.2.2, était opérationnel mais ne s'utilisait pas de façon transparente. Il n'était pas interopérable avec d'autres ORB, était utilisable par seulement deux nœuds, et avait une procédure d'initialisation particulière. Il était suffisant pour effectuer des mesures de performances en ping-pong, mais insuffisant pour une utilisation normale. Nous présentons dans la suite ce qui a été réalisé pour que notre prototype haute performance devienne une implémentation complète qui s'utilise de façon entièrement transparente.

4.4 Approche retenue pour l'implémentation complète : socket virtuelles

Nous cherchons à modifier l'ORB aussi peu que possible, de façon à pouvoir suivre facilement l'évolution des versions. Il est en effet illusoire de maintenir simultanément

plusieurs versions d'un même ORB. C'est pourquoi nous ne modifions que les couches de transport et de multi-threading.

OmniORB utilise une classe de portabilité pour abstraire les processus légers. L'adaptation de cette classe aux threads Marcel [10] est réalisée très facilement car Marcel implémente entièrement le sous-ensemble de l'API des threads Posix qu'OmniORB utilise. Marcel est une bibliothèque de multi-threading de niveau utilisateur qui garantit une bonne réactivité par rapport au réseau quand on l'utilise conjointement à Madeleine.

En ce qui concerne la couche de transport, nous choisissons de modifier OmniORB au minimum. Le protocole TCP/IP, conçu pour les réseaux longues distances, n'est pas adapté à une utilisation haute performance sur une grappe. Cependant, l'interface classique des sockets Berkeley convient. Nous choisissons donc d'utiliser les services de Madeleine au travers d'une API de type socket, en implémentant des sockets virtuelles que nous appelons `VSock`. Nous suivons le schéma de la figure 6.

`VSock` implémente un sous-ensemble de l'API standard des sockets. Il est basé sur Madeleine. Il a une démarche similaire à Fast Sockets [28] construit au-dessus de la bibliothèque de communications Active Messages. Dans un but de haute performance, `VSock` ne fournit qu'un transport de datagrammes en mode "zéro copie". `VSock` ne supporte pas les flux continus tels que TCP les fournit habituellement ; ceci n'est pas gênant si le logiciel qui utilise `VSock` gère convenablement les frontières des messages, et cela permet de n'avoir aucune copie supplémentaire. Ces services suffisent pour OmniORB et MICO. Les mécanismes d'établissement de connexion sont exactement ceux des sockets, et `VSock` utilise l'adressage IP standard. De cette façon, porter OmniORB sur `VSock` est réalisé simplement en détournant les appels des primitives socket standard vers leur équivalent `VSock`. L'adaptation d'OmniORB à `VSock` est réalisée par un simple script agissant sur les sources d'OmniORB.

Cette approche conserve le protocole GIOP comme format de message. Le transport de ces messages est réalisé par Madeleine. De cette façon, la gestion des différents protocoles est réalisée au niveau bas, sous GIOP. L'utilisation de GIOP par rapport à l'utilisation d'un ESIOP (voir la fin de la section 2.1) n'est probablement pas la plus performante à l'exécution du point de vue de la latence ; cependant, au vu des résultats des mesures de performance, nous considérons que dans OmniORB le surcoût logiciel de GIOP est faible et ne justifie pas pour l'instant le développement d'un ESIOP.

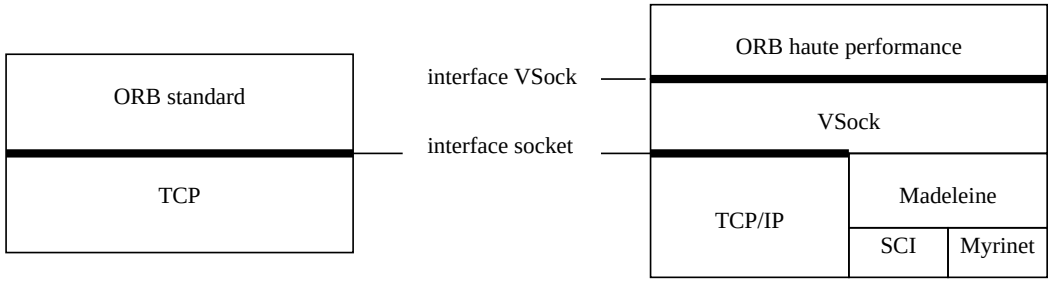

FIG. 6 – *Vue d'ensemble de l'implémentation* CORBA *haute performance*

4.5 Allocation mémoire à la réception

La méthode d'allocation de la mémoire à la réception est un autre facteur déterminant pour obtenir des hauts débits. Ne faire aucune copie intermédiaire n'est pas suffisant pour exploiter entièrement le débit permis par les réseaux haut débit. Le cas le plus marquant est celui de Myrinet-2000, de débit nominal 250 Mo/s, où l'allocation mémoire standard a un effet dramatique sur les performances. Quand il reçoit une requête, OmniORB alloue un espace mémoire pour stocker les paramètres. Cette allocation est réalisée au moyen de l'opérateur C++ standard `new`. Comme nous le voyons sur la figure 7, avec cette méthode, le débit plafonne à seulement 130 Mo/s et a un comportement totalement erratique avec de gros messages. Ceci a été constaté particulièrement avec BIP sur les réseaux Myrinet et Myrinet-2000.

Pour isoler le problème, nous réalisons deux benchmarks sur des Pentium III 1 GHz : l'un effectue une copie de mémoire à mémoire avec la destination fraîchement allouée par `malloc`, l'autre effectue un transfert par `VSock` avec une allocation par `malloc` à la réception. Le premier benchmark montre que même quand la destination est fraîchement allouée, le débit obtenu est supérieur à 240 Mo/s. Le deuxième benchmark (présenté sur la figure 7) montre un comportement erratique pour les messages au-delà de 64 ko, similaire à OmniORB et l'allocateur `new` standard. Ceci est dû à la stratégie de `malloc` : au-delà d'un certain seuil (64 ko sous Linux, glibc 2.1), il alloue un nouveau segment de mémoire à l'aide de la primitive `mmap` ; il s'ensuit des défauts de pages en cascade quand on écrit dans cette zone pour la première fois. Ces défauts de pages en cascade n'empêchent pas un accès mémoire (copie de mémoire à mémoire par exemple) à plus de 240 Mo/s. Cependant, quand ils ont lieu en même temps qu'une réception réseau par DMA (Direct Memory Access), l'entrelacement des deux opérations provoque une grande perte de débit. Cet effet est très visible

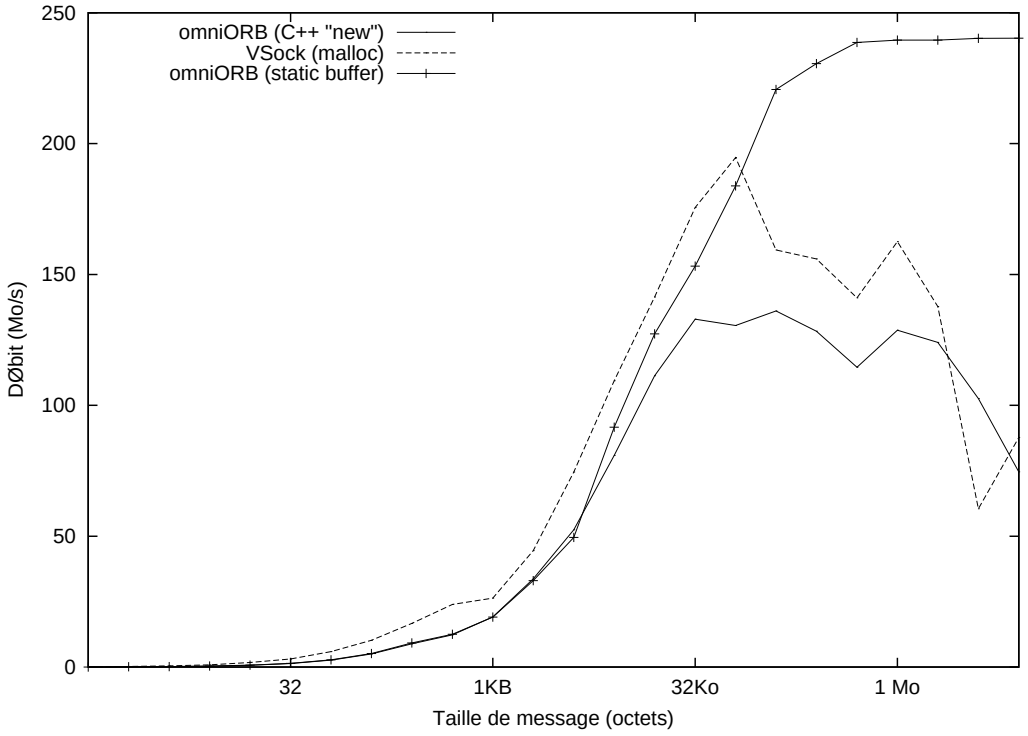

FIG. 7 – *Débits mesurés sur Myrinet-2000 en fonction de l'allocateur de mémoire*

sur les réseaux Myrinet et Myrinet-2000, et dans une moindre mesure sur les réseaux SCI. Le facteur limitant est donc bien la réception depuis le réseau vers une zone mémoire allouée dynamiquement.

Nous réalisons un allocateur optimisé pour une réception rapide. Il utilise un tampon statique dans lequel nous allouons dynamiquement des zones pour la réception des paramètres avec une simple stratégie *round-robin*. Puisque ces zones sont dans un tampon statique, elles sont déjà allouées par le système d'exploitation au moment de la réception d'un message ; aucun défaut de page ne se produit si cette zone n'a pas été paginée vers le disque. Grâce à cet allocateur de blocs dans un tampon statique, nous obtenons la courbe désignée par "static buffer" de la figure 7 : OmniORB atteint le débit de 240 Mo/s et son comportement est beaucoup plus régulier. Ce débit est le maximum utilisable par Madeleine sur Myrinet-2000, ce qui représente 96 % du débit nominal du matériel.

4.6 Interopérabilité

Nous portons une attention toute particulière à l'interopérabilité de notre implémentation CORBA haute performance avec les autres implémentations CORBA. L'interopérabilité avec des ORB standard (sans Madeleine) est absolument nécessaire. Ceci implique que `VSock` soit transparent de trois façons :

- Sélection automatique du protocole : du point de vue de l'application, l'ORB haute performance doit se comporter comme un ORB standard. L'application n'a pas besoin de savoir (et ne *ne doit pas* savoir) quel est le protocole réseau sous-jacent. C'est à `VSock` que revient la tâche de sélectionner le protocole réseau adapté à chaque lien, en fonction du matériel disponible. `VSock` doit être capable de décider automatiquement si un objet est sur un nœud accessible par Madeleine ou s'il faut se contenter de TCP.
- Transparence IIOP : IIOP est le protocole standard utilisé par les ORB. IIOP est l'incarnation de GIOP sur TCP/IP. IIOP spécifie l'intégralité du protocole, du format des données aux en-têtes des messages. Pour que l'ORB soit interopérable, il est donc nécessaire que le protocole de `VSock` sur TCP/IP soit complètement transparent, c'est-à-dire que du point de vue de l'application `VSock` sur TCP est exactement TCP. `VSock` ne doit ajouter aucun en-tête lors d'une utilisation sur TCP, pas même au moment de la connexion pour sélectionner le protocole.
- Mapping des adresses : Madeleine utilise un adressage par numéros logiques (comme MPI), alors que IIOP utilise le système d'adresses IP. L'interopérabilité impose [1] que les objets soient toujours décrits par des adresses IP et que `VSock` réalise lui-même la résolution des adresses IP en adresse Madeleine.

La stratégie retenue dans `VSock` pour gérer TCP et Madeleine en même temps de façon transparente est la suivante : un client résout l'adresse IP en numéro Madeleine grâce à une table inverse. Si la résolution d'adresse échoue, cela signifie que l'objet est à l'extérieur de la grappe, et donc `VSock` utilise le protocole TCP. Si la résolution réussit, il reste encore à vérifier que l'objet visé est effectivement géré par Madeleine– ce n'est pas parce qu'une machine est dans une grappe Madeleine que tous ses processus et donc tous ses objets sont gérés par Madeleine ! Nous identifions les liens `VSock` par le port TCP/IP associé. Savoir si un objet est atteignable par Madeleine revient à savoir si `VSock` connaît la combinaison adresse IP-port.

1. Même si la norme prévoit que les références d'objets (IOR) peuvent contenir plusieurs types d'adresses pour plusieurs protocoles, il existe des ORB qui sont incapables d'utiliser plus d'une adresse. OmniORB en fait partie.

4.7 Client/serveur et SPMD : gestion de la dynamicité

CORBA est orienté client/serveur, et a une topologie réseau dynamique. Des clients CORBA sont lancés et se connectent dynamiquement à des serveurs. À l'opposé, Madeleine a une topologie réseau statique. Comme la plupart des bibliothèques de communications pour grappes, l'approche est SPMD (*Single Program Multiple Data*). Ceci signifie que les programmes sur les différents nœuds doivent être identiques et démarrés en même temps.

Ces deux approches sont diamétralement opposées *a priori*. Notre but est donc de projeter une topologie dynamique client/serveur sur une topologie statique SPMD. Cette tâche n'est pas du ressort d'une implémentation CORBA ; nous avons choisi de la confier à l'exécutif PadicoTM [12].

La solution adoptée par PadicoTM est de lancer un programme de *bootstrap* identique sur chaque nœud susceptible de recevoir un processus – client ou serveur CORBA. De cette façon, la contrainte "SPMD" de Madeleine est satisfaite. Ensuite, ce programme charge dynamiquement les binaires des clients et serveurs CORBA sous forme de bibliothèques dynamiques (ce sont les "`.so`" sous Unix). Grâce à ce mécanisme, des binaires différents peuvent être chargés bien que le démarrage soit SPMD, et des connexions réseaux sont établies et coupées en cours de session bien que la topologie globale soit statique.

4.8 Utilisation simultanée de CORBA et MPI

Nous visons les applications de couplage de codes basées sur les objets CORBA parallèles [11]. Typiquement, ces applications utilisent simultanément une interface CORBA pour les communications externes et une interface MPI pour les communications internes. L'utilisation d'un réseau rapide ou d'une bibliothèque de multi-threading par plusieurs exécutifs en même temps pose problème si les exécutifs ne tiennent pas compte les uns des autres. Par exemple, certaines bibliothèques de communication n'autorisent qu'un seul client (BIP sur Myrinet, par exemple), d'autres arriveraient rapidement à court de ressources si chaque exécutif gérait séparément ses canaux de communication (sur SCI par exemple). À l'intérieur d'un même processus, chaque exécutif doit utiliser la même bibliothèque de multi-threading pour qu'il n'y ait pas de conflit. Enfin, il est souhaitable que les accès aux ressources soit gérés de manière coopérative plutôt que concurrente.

Pour gérer ces problèmes spécifiques à la cohabitation de plusieurs exécutifs haute performance, nous utilisons PadicoTM [13] (ou *Padico Task Manager*) déjà utilisé pour gérer la dynamicité. Son rôle est de fournir un cadre haute performance permet-

tant l'intégration de plusieurs exécutifs communiquants. PadicoTM est l'exécutif de l'environnement Padico, une plate-forme de recherche qui intègre le calcul parallèle, le calcul distribué, et les composants logiciels.

PadicoTM enrichit Madeleine de fonctionnalités de multiplexage des différents messages. Les performances des communications ont été préservées grâce à la mise en œuvre de mécanismes de gestion des en-têtes et de callbacks. Un mécanisme efficace de gestion des en-têtes permet aux couches logicielles d'ajouter leurs en-têtes sans augmenter le nombre de messages sur le réseau. Ainsi, la latence n'est pas pénalisée par le multiplexage et les différentes couches. Pour éviter que chaque exécutif (CORBA, MPI, etc.) mettent en place une attente réseau concurrente des autres, PadicoTM propose un mécanisme de *callbacks*: chaque exécutif enregistre la fonction qu'il souhaite voir appeler lorsqu'un message arrive sur son canal de communication logique. PadicoTM gère le thread de réception des communications Madeleine. Grâce à ces mécanismes, les performances de l'implémentation CORBA ainsi que l'implémentation MPI basée sur MPICH/Madeleine [6] ne sont pas changées par rapport à une implémentation sans PadicoTM.

4.9 Performances CORBA

4.9.1 Mesures

La configuration de test est composée de machines bi-processeur Pentium II 450 Mhz équipées de cartes Ethernet-100, SCI et Myrinet, ainsi que de machines bi-Pentium III 1 GHz équipées de cartes Myrinet-2000. La mesure de performance consiste en l'invocation d'une méthode distante qui accepte un paramètre de longueur variable (séquence de `Long` en mode *inout*), comme décrit à la section 3.1. Nous mesurons la latence et le débit.

Le débit obtenu par notre implémentation CORBA haute performance est présenté à la figure 8. Le débit maximum mesuré est de 86 Mo/s sur SCI, 101 Mo/s sur Myrinet et 240 Mo/s sur Myrinet-2000 (soit près de 2 Gbit/s). Ces performances sont excellentes : nous exploitons plus de 99 % du débit atteint par Madeleine.

La latence est de 160 μs sur TCP/Ethernet-100 avec l'implémentation OmniORB de base. Sur les réseaux SCI et Myrinet avec OmniORB sur `VSock`, la latence descend à 55 μs. Sur les machines rapides dotée de Myrinet-2000, elle descend à 20 μs. Ce sont les meilleures performances obtenues par une implémentation CORBA à notre connaissance.

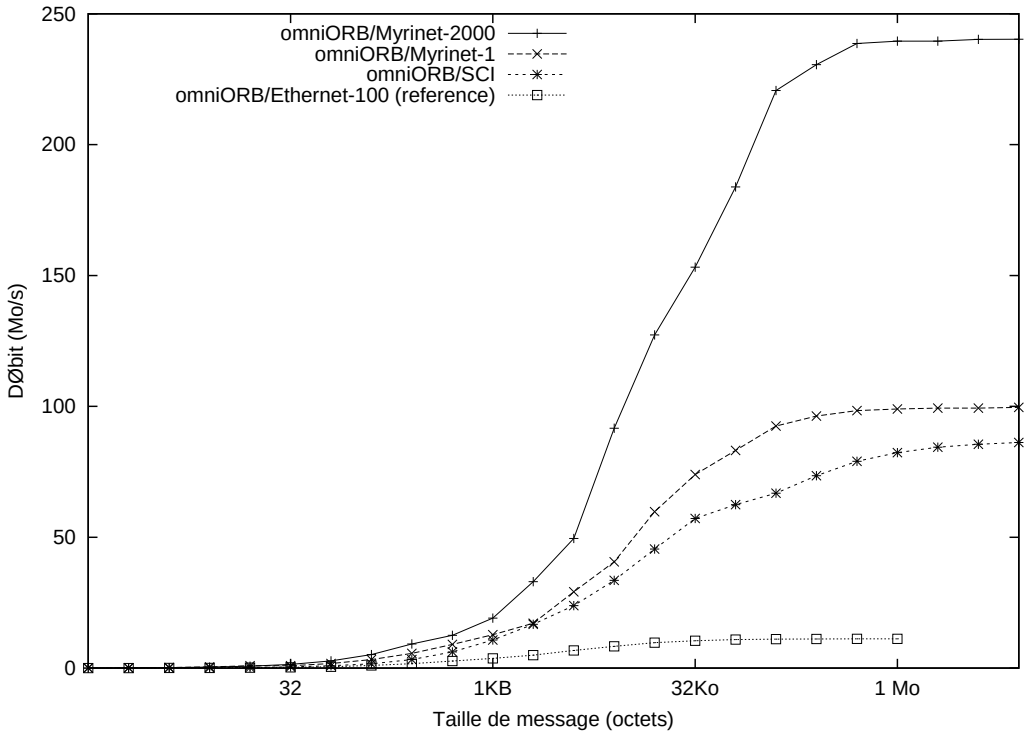

FIG. 8 – *Débit d'OmniORB sur les réseaux rapides*

4.9.2 Comparaison avec Mpi

Le principal intérêt de l'utilisation de Corba dans le cadre d'applications numériques est de permettre une meilleure structuration des applications. Il est capital que cela ne se traduise pas par une dégradation des performances ! Notons que grâce à PadicoTM, Mpi et Corba sont utilisables en même temps.

La figure 9 représente le débit respectif de Corba (OmniORB/`VSock`) et Mpi (MPICH/Madeleine [6]) sur le réseau Myrinet-2000. On observe que les deux courbes sont très proches : Corba et Mpi font jeu égal en ce qui concerne le débit. Nous obtenons le même type de courbes sur Sci et Myrinet. Ce résultat n'est pas surprenant dans la mesure où OmniORB utilise une approche similaire à celle de MPICH/Madeleine : des transferts sans copie en utilisant Madeleine pour l'accès au réseau.

La latence de Mpi est de 23 μs sur Sci et Myrinet, 11 μs sur Myrinet-2000, à comparer aux 55 μs et 20 μs de Corba respectivement sur ces réseaux. Cette différence n'est toutefois pas gênante. Dans le cas des applications de couplage de codes que nous visons, la latence n'est pas critique car ces applications communiquent de grands volumes de données. Cependant, si la latence s'avérait être importante, nous pourrions l'améliorer notablement. Notre implémentation Corba haute performance est basée sur le protocole Giop qui est très coûteux et partiellement inutile en environnement homogène. La norme Corba prévoit l'utilisation d'autres protocoles de transport (appelés Esiop) spécifiques. Pour diminuer la latence, il serait possible d'écrire un Esiop spécifique au-dessus de Madeleine, comme par exemple LGIOP [24], un Esiop dérivé de Giop développé pour Tao.

Cependant, nous jugeons ces chiffres de latence suffisants pour l'instant. Nous estimons qu'il est plus judicieux de concentrer nos efforts sur d'autres points pour le moment, comme par exemple le déploiement des applications et l'allocation des ressources.

5 Travaux connexes

Dès les débuts de Corba, des travaux ont été réalisés pour améliorer ses performances. Il existe ainsi un certain nombre d'implémentations sur des réseaux haut débit. Ces implémentations sont expérimentales, et ne sont généralement pas dif-

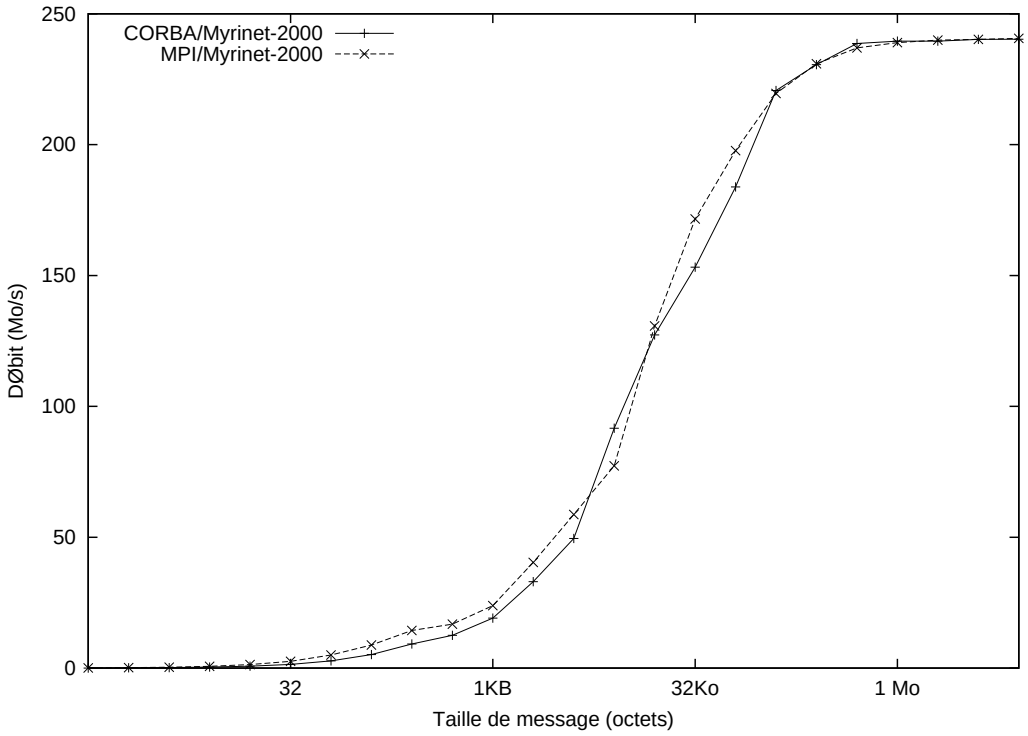

FIG. 9 – *Comparaison du débit de* CORBA *et* MPI *sur Myrinet-2000*

fusées. Aucun des travaux n'a véritablement abouti. Nous en dressons ici une liste représentative, quoique non exhaustive.

- OmniORB 2 : dans sa version 2 (1998), OmniORB avait été adapté aux réseaux ATM et SCI. Le code correspondant n'étant pas distribué, nous rapportons uniquement les résultats publiés [21, 25].
  Sur ATM, la latence de 360 μs est acceptable compte tenu des machines utilisées [25]. Le débit de communication monte à 17 Mo/s, soit 135 Mbit/s pour un lien physique à 155 Mbit/s, ce qui est une bonne performance. Cependant, ce débit est obtenu uniquement avec des blocs d'octets non structurés, donc sans coût d'encodage GIOP. Pour un type structuré, il tombe en-dessous de 6 Mbit/s ce qui est très faible.
  Avec une latence de 156 μs et un débit jusqu'à 37 Mo/s (300 Mbit/s), les performances annoncées sur SCI sont relativement bonnes [21]. Cependant, les performances obtenues avec des types structurés ne sont pas présentées. Au vu des résultats de la version ATM, nous supposons que le débit pour des types structurés est beaucoup plus faible que les 37 Mo/s annoncés pour les blocs d'octets.
  Dans cette version 2, OmniORB n'était pas très optimisé et réalisait en particulier beaucoup de copies en mémoire. La version 3 est beaucoup plus performante, mais les versions destinées aux réseaux haut débit ont été abandonnées.
- MICO/VIA : un portage de MICO sur VIA [15] a été réalisé par Nhu-Tung Doan [14]. Cette mise en œuvre a un débit jusqu'à 12 % plus rapide qu'une utilisation du même matériel Gigabit Ethernet avec TCP. Par contre, il n'est fait aucune mention de la latence.
- CrispOrb [19] est développé par les laboratoires Fujitsu. Il est destiné spécifiquement aux réseaux VIA, et a été testé sur SC-net, une émulation logicielle de VIA sur le réseau haute performance Fujitsu Synfinity-0. Le gain en latence est de 20 % par rapport à une utilisation du même matériel par TCP/IP.
- TAO [20] a été porté sur ATM. Cependant, l'accent était mis essentiellement sur les aspects temps réel et qualité de service plutôt que sur la haute performance.

Les résultats des travaux existants sont encourageants mais semblent très incomplets. Chaque implémentation est spécifique à un type de réseau particulier ce qui restreint l'utilisation potentielle. Par exemple aucune ne supporte le réseau Myrinet. De plus, ce sont des orientations marginales : le développement d'OmniORB sur les réseaux rapides a été arrêté après un prototype confidentiel, le prototype MICO/VIA n'a pas abouti et CrispOrb cible une machine en particulier. Aucun de ces travaux

n'a pour objectif de mener à une implémentation CORBA utilisable sur les grappes de calculs, en tenant compte de la diversité des réseaux (Myrinet, SCI, VIA, etc.)

Nous nous démarquons également de TAO qui privilégie avant tout l'aspect temps réel avec RT-CORBA. TAO cherche à délivrer des performances garanties, reproductibles, et n'a pas été porté à notre connaissance sur des réseaux rapides autres qu'ATM. Au contraire, nous cherchons la meilleure performance possible *en moyenne*, en restant dans l'optique "best effort". Nous implémentons CORBA sans extension temps réel.

6 Conclusion

Une grille de calcul est un environnement hétérogène, en particulier en ce qui concerne les infrastructures et les protocoles réseaux. Pour obtenir de bonnes performances, il n'est pas raisonnable de considérer tous les liens comme équivalents au niveau des modèles de programmation. Il est préférable que les applications voient la hiérarchie du réseau. Un code parallèle est géré par exemple par MPI, qui est déployé sur une grappe ou une machine parallèle, et profite d'une faible latence et d'un haut débit. Ces codes parallèles sont couplés par CORBA. Si deux codes couplés par CORBA sont déployés sur des sites distants, il est nécessaire d'utiliser TCP/IP. Si ces codes sont déployés sur la même grappe, il est souhaitable que CORBA soit capable d'utiliser le réseau rapide de la grappe. Notre objectif est que le même code puisse être déployé aussi bien sur plusieurs sites distants que sur une seule grande grappe, de façon entièrement transparente. Il est donc souhaitable de disposer d'une implémentation CORBA capable de s'adapter selon le cas à une communication sur WAN et LAN par TCP/IP ou sur un SAN tel que Myrinet ou SCI.

Nous avons analysé les implémentations CORBA actuelles dans le cadre d'une utilisation sur une grappe de PC. Nous avons modélisé leurs performances, puis implémenté deux prototypes sur la bibliothèque de communications Madeleine, pour guider notre choix final. Nous avons ensuite réalisé une implémentation complète d'un ORB haute performance en nous basant sur OmniORB et Madeleine. Cet ORB haute performance est interopérable avec les ORB classiques et s'utilise de façon transparente. Enfin, nous avons mesuré ses performances et montré qu'elles sont comparables à ce que MPI obtient dans les même conditions.

Les résultats obtenus sont excellents. La réputation de mauvaises performances de CORBA n'est pas fondée sur un aspect intrinsèque du modèle CORBA, mais sur des implémentations qui ne se souciaient pas des performances. Cette étude a débouché sur une implémentation opérationnelle d'un ORB haute performance capable

d'exploiter entre autres les réseaux SCI, Myrinet et Myrinet-2000, obtenant un débit de 240 Mo/s soit près de 2 Gbit/s sur ce dernier réseau.

Grâce à ces résultats, notre choix de CORBA comme moyen de couplage des applications de calcul numérique sur SAN aussi bien que sur LAN et WAN est validé. Notre approche de couplage de codes parallèles par CORBA soulève d'autres problèmes qui sont au-delà de la portée de cet article et ont été résolus. Il a fallu définir la notion d'objet CORBA parallèle [27, 11]. Il a également été nécessaire de concevoir un exécutif capable de faire fonctionner MPI et CORBA en même temps, les deux accédant aux ressources de façon coopérative grâce à PadicoTM [12, 13]. Il reste encore à résoudre les problèmes liés au déploiement des applications et à l'allocation des ressources qui seront également pris en charge par l'environnement Padico.

Remerciements

L'auteur tient à remercier la « PM2 team » du LIP (ENS Lyon), particulièrement Olivier Aumage et Raymond Namyst, pour leur assistance rapide et efficace, ainsi que pour avoir adapté Madeleine et Marcel à ses besoins. Il remercie également Christian Pérez (IRISA, Rennes) pour ses suggestions et ses relectures de cet article.

Références

[1] MICO, an OpenSource CORBA implementation. `http://www.mico.org`.

[2] ORBacus(tm) for C++ and Java. World Wide Web document, `http://www.orbacus.com/products/orbacus.html`.

[3] PM2 High Perf. World Wide Web document, `http://www.pm2.org`.

[4] The Object Management Group. World Wide Web document, `http://www.omg.org`.

[5] OmniORB Home Page. AT&T Laboratories Cambridge, `http://www.omniorb.org`.

[6] O. Aumage, G. Mercier, and R. Namyst. MPICH/Madeleine: a true multi-protocol MPI for high-performance networks. In *Proc. 15th International Parallel and Distributed Processing Symposium (IPDPS 2001)*, page 51, San Francisco, April 2001. IEEE.

[7] Olivier Aumage, Luc Bougé, Alexandre Denis, Jean-François Méhaut, Guillaume Mercier, Raymond Namyst, and Loïc Prylli. A portable and efficient communication library for high-performance cluster computing. In *IEEE Intl Conf.*

on Cluster Computing (CLUSTER 2000), pages 78–87, Technische Universität Chemnitz, Saxony, Germany, November 2000.

[8] Olivier Aumage, Luc Bougé, Lionel Eyraud, and Raymond Namyst. Communications efficaces au sein d'une interconnexion hétérogène de grappes : Exemple de mise en oeuvre dans la bibliothèque Madeleine. Soumis pour publication à *Calculateurs parallèles*. Numéro spécial Métacomputing: calcul réparti à grande échelle, March 2001.

[9] N. J. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. S., and W.-K. Su. Myrinet : A gigabit-per-second local area network. *IEEE-Micro*, 15(1):29–36, February 1995.

[10] V. Danjean, R. Namyst, and R. Russell. Integrating kernel activations in a multithreaded runtime system on Linux. In *Parallel and Distributed Processing. Proc. 4th Workshop on Runtime Systems for Parallel Programming (RTSPP '00)*, volume 1800 of *Lect. Notes in Comp. Science*, pages 1160–1167, Cancun, Mexico, May 2000. In conjunction with *IPDPS 2000. IEEE TCPP and ACM*, Springer-Verlag.

[11] A. Denis, C. Pérez, and T. Priol. Portable parallel corba objects: an approach to combine parallel and distributed programming for grid computing. In *Proc. of the 7th Intl. Euro-Par'01 conf.*, pages 835–844, Manchester, UK, August 2001. Springer.

[12] A. Denis, C. Pérez, and T. Priol. Towards high performance CORBA and MPI middlewares for grid computing. In Graig A. Lee, editor, *Proc of the 2nd International Workshop on Grid Computing*, number 2242 in LNCS, pages 14–25, Denver, Colorado, USA, November 2001. Springer-Verlag. In conjunction with *SuperComputing 2001 (SC'01)*.

[13] A. Denis, C. Pérez, and T. Priol. PadicoTM: An open integration framework for communication middleware and runtimes. In *IEEE International Symposium on Cluster Computing and the Grid (CCGRID2002)*, 2002. à paraître.

[14] Nhu-Tung Doan. Optimizing inter-ORB communication for a high-performance distributed object broker. Master's thesis, IRISA, October 2000.

[15] Dave Dunning, Greg Regnier, Gary McAlpine, Don Cameron, Bill Shubert, Frank Berry, Anne-Marie Meritt, Ed Gronke, and Chris Dodd. The Virtual Interface Architecture. *IEEE Micro*, pages 66–75, March 1998.

[16] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.

[17] I. Foster and C. Kesselman, editors. *The Grid: Blueprint for a New Computing Infracstructure*. Morgan Kaufmann Publishers, Inc, 1998.

[18] IEEE. Standard for Scalable Coherent Interface (SCI). Standard no. 1596, August 1993.

[19] Yuji Imai, Toshiaki Saeki, Tooru Ishizaki, and Mitsushiro Kishimoto. CrispORB: High performance CORBA for system area network. In *Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing*, pages 11–18, 1999.

[20] Fred Kuhns, Douglas Schmidt, and David Levine. The design and performance of a real-time I/O subsystem. In *Proceedings of the 5th IEEE Real-Time Technology and Applications Symposium (RTAS99)*, Vancouver, Canada, June 1999.

[21] Sai-Lai Lo and Steve Pope. The implementation of a high performance ORB over multiple network transports. rapport de recherche, Olivetti & Oracle Laboratory, Cambridge, March 1998.

[22] OMG. Benchmark PSIG, White Paper on Benchmarking. OMG Document bench/99-12-01, December 1999.

[23] OMG. The Common Object Request Broker: Architecture and Specification (Revision 2.5). OMG Document formal/01-09-34, September 2001.

[24] C. O'Ryan, F. Kuhns, D. C. Schmidt, and J. Parsons. *Design Patterns in Communications*, chapter Applying Patterns to Develop a Pluggable Protocols Framework for ORB Middleware. Cambridge University Press, 2000.

[25] Steve Pope and Sai-Lai Lo. The implementation of a native ATM transport for a high performance ORB. rapport de recherche, Olivetti & Oracle Laboratory, Cambridge, June 1998.

[26] L. Prylli and B. Tourancheau. Bip: a new protocol designed for high performance networking on myrinet. In *1st Workshop on Personal Computer based Networks Of Workstations (PC-NOW '98)*, Lect. Notes in Comp. Science, pages 472–485. Springer-Verlag, apr 1998. In conjunction with *IPPS/SPDP 1998*.

[27] C. René and T. Priol. MPI code encapsulating using parallel CORBA object. In *Proceedings of the Eighth IEEE International Symposium on High Performance Distributed Computing (HPDC '99)*, pages 3–10, August 1999.

[28] Steven H. Rodrigues, Thomas E. Anderson, and David E. Culler. High-performance local area communication with fast sockets. In *USENIX '97*, pages 257–274, January 1997.

[29] Douglas Schmidt, Aniruddha Gokale, Timothy Harrison, and Guru Parulkar. A High-performance Endsystem Architecture for Real-time CORBA. *IEEE Communication Magazine*, 14(2), February 1997.

Unité de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unité de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
http://www.inria.fr
ISSN 0249-6399