Towards High Performance CORBA and MPI Middlewares for Grid Computing

Alexandre Denis, Christian Pérez, Thierry Priol

To cite this version:

Alexandre Denis, Christian Pérez, Thierry Priol. Towards High Performance CORBA and MPI Middlewares for Grid Computing. 2nd International Workshop on Grid Computing, Nov 2001, Denver/USA, United States. pp.14-25. inria-00000130

HAL Id: inria-00000130
https://hal.inria.fr/inria-00000130

Submitted on 24 Jun 2005

Towards High Performance CORBA and MPI Middlewares for Grid Computing

Alexandre Denis[1], Christian Pérez[2], and Thierry Priol[2]

[1]IRISA/IFSIC, [2]IRISA/INRIA,

Campus de Beaulieu - 35042 Rennes Cedex, France
{Alexandre.Denis,Christian.Perez,Thierry.Priol}@irisa.fr

Abstract. Due to the high level of heterogeneity in a computational Grid, designing a runtime system for such computing infrastructure is extremely challenging, for example regarding the ability to exploit transparently and efficiently various networking technologies. Programming a computational Grid often requires the use of several communication paradigms (RPC, RMI, DSM, Message passing) that have to share these networking resources. This paper presents the first step towards a runtime system that allows efficient communication for various communication-oriented middlewares. We introduce a CORBA implementation that reaches 240 MB/s, which is as far as we know the best CORBA performance. Thus, CORBA can be as efficient as MPI on high performance networks. Moreover, we show that different communication middlewares, like CORBA and MPI, can efficiently co-habit within the same runtime system taking full benefit of various networking resources (SAN to WAN).

1 Programming the Grid

Due to the high level of heterogeneity in a computational Grid, designing a runtime system for such computing infrastructure is extremely challenging. In this paper we focus on a particular facet that a grid runtime has to tackle: managing various communication resources and hiding them so that middlewares can use them transparently and efficiently.

Beside various communication technologies, the design of grid applications requires different middlewares allowing programmers to use programming models that are most suitable for their applications. Although first implementations of Grid infrastructures, such as Globus[8], support mainly the execution of message-based applications, it is foreseen that future grid applications will require much more advanced programming models based on either distributed objects or components. Among such grid applications, multi-physics applications are good examples. They are made of several high-performance simulation codes coupled together to simulate several physics behaviors. Each phenomenon is simulated by a parallel simulation code. This kind of application appears well suited for the Grid because many of its codes need either a parallel machine or a vector supercomputer to run in order to keep the computation time within reasonable bounds. The codes that compose a coupling application are generally independently developed. It appears very constraining to require that all codes are based on the same communication paradigm, like for example MPI to be able to run on a computational grid. We advocate an approach that lets the application designer choose the most suitable communication paradigm. Within a parallel code, it may be MPI, PVM, a distributed shared memory system (DSM), a parallel language like OpenMP[7], etc. The coupling of the simulation codes could be carried out through the use of a Remote Method Invocation mechanism (Java RMI or CORBA) to transfer the control between the simulation codes.

Such an approach requires several communication middlewares to exploit various networking technologies. Depending on the computing resource availability, several simulation codes could be mapped onto a WAN or onto the same parallel machine. In the later case, the RMI mechanism should be able to exploit the underlying network of a parallel machine. Current implementations of existing RMIs (Java RMI or CORBA) do not support such specific network so that the coupling application cannot fully exploit the communication resources.

In this paper, we advocate the choice of the CORBA technology to couple simulation codes. CORBA has some very interesting features. It has been designed for distributed communication. So, it harnesses adequately the heterogeneity of the different computers. Moreover, it offers an object oriented framework. Last, it offers binding for most languages[1]. CORBA has to fulfill two important requirements: efficiency on high speed networks, and interconnect two parallel codes. This paper aims at giving a positive answer to the performance of CORBA on high speed networks.

The answer to the second requirement is twofold. First, the OMG[2] has issued an RFP[14] (Request For Proposal) that solicits proposals to extend CORBA functionality to conveniently and efficiently support parallel processing applications. A response[13] was submitted by a consortium of several industrial companies and a supporting organization. The proposed approach shares some similarities with previous works [10, 16]. Second, we are working on providing similar functionalities – i.e. CORBA support for parallel applications – but based on standard CORBA 2 [6]. Our motivation is that normalization is a long process and it is not clear whether most ORB will implement it.

The remainder of this paper is divided as follows. Section 2 presents the challenges that our approach has to face. In section 3, the first challenge, a high performance CORBA, is overcome. Our second challenge, concurrent support of several middlewares, is the subject of section 4. All these results are gathered in a coherent platform Padico which is sketched in section 5. Then we conclude in section 6.

[1] The mapping to FORTRAN9x is not official but a study that has been carried out within the Esprit PACHA project has shown that such a mapping is possible

[2] Object Management Group – the consortium that defines CORBA

2 Communication Issues in a Grid Environment

2.1 Grid Infrastructures

Grid computing infrastructures cover a wide range of machines, going from supercomputer to cluster of workstations. While the former is still a platform of choice for computing-intensive applications, the success of the latter is always growing due to their competitive performance/price ratio. A grid computing middleware must be portable enough to run on every machine of the grid.

The Grid is composed of several kinds of networks: SAN on clusters of workstations (eg. Myrinet, SCI, VIA), dedicated interconnection networks on supercomputers, and WAN. Multi-threading is more and more required by middlewares like MPI or CORBA. Also, it is an efficient paradigm to support concurrently several middlewares. So, it is challenging to design a grid computing runtime system that is both *portable* and *efficient*.

2.2 CORBA

As CORBA is a corner stone of our approach, it is critical to have a high performance CORBA implementation (ORB) able to exploit various networking technologies (from dedicated networks within supercomputers to SAN). However, such an implementation must overcome some challenges.

A high performance CORBA implementation will typically utilize SAN with a dedicated high-performance protocol. It needs to be interoperable with other standard ORBs, and thus should implement both high-speed protocol for SAN and standard IIOP (Internet Inter-Orb Protocol) for interconnecting with other ORBs over TCP/IP. From the application, the high-speed ORB must behave as any other ORB. We aim at using standard CORBA applications on our high-performance ORB. Network adapter selection, protocol selection and address resolution must be automatic and fully hidden.

There is a network model discrepancy between the "distributed world" (eg. CORBA) and the "parallel world" (eg. MPI). Communication layers dedicated to parallelism typically use a static topology[3]: nodes cannot be inserted or removed into the communicator while a session is active. On the other hand, CORBA has a distributed approach: servers may be dynamically started, clients may dynamically contact servers. The network topology is dynamic. It is challenging to map the distributed communication model onto SAN that are biased toward the parallel communication model.

2.3 Supporting Several Middlewares at the Same Time

Supporting CORBA and MPI, *both running simultaneously*, is not not as straightforward as it may seem. Several access conflicts for networking resources

[3] PVM and MPI2 address this problem but do not allow network management on a link-per-link basis.

may arise. For example, only one application at a time can use Myrinet through BIP [17]. If both CORBA and MPI try to use it without being aware of each other, there are access conflicts and reentrance issues. If each middleware (eg. CORBA, MPI, a DSM, etc.) has its own thread dedicated to communications, with its own policy, communication performance is likely to be sub-optimal. In a more general manner, resource access should be cooperative rather than competitive.

2.4 Madeleine and Marcel

To face the heterogeneity of the Grid, a portability layer for network and multi-threading management should be adopted. At first look, it may seem attractive to use a combination of MPI and PosixThreads as foundation. However, [4] shows that this solution has drawbacks. To deal with portability as well as low level issues, we choose the Madeleine communication layer [2] and the Marcel multi-threading library [5]. The Madeleine communication layer was designed to bridge the gap between low-level communication interfaces (such as BIP [17], SBP or UNET) and middlewares. It provides an interface optimized for *RPC-like* operations that allows zero-copy data transmissions on high-speed networks such as Myrinet or SCI. Marcel is a multi-threading library in user space. It implements an N:M thread scheduling on SMP architectures. When used in conjunction with Marcel, Madeleine is able to guarantee a good reactivity of the application to network I/O.

3 High Performance CORBA

3.1 Related Works

Previous works have already be done about high performance CORBA. TAO [11] (the ACE ORB) focuses on high performance and real-time aspects. Its main concern is predictability. It may utilize TCP or ATM networks, but it is not targeted to high performance network protocols found on clusters of PCs such as BIP or SISCI. OmniORB2 had been adapted to ATM and SCI networks. Since the code is not publicly available, we only report published results. On ATM, there is a gap of bandwidth between raw bytes and structured data types [15]. The bandwidth can be as low as 0.75 MB/s for structured types. On SCI, results are quite good [12] (156 μs, 37.5 MB/s) for messages of raw bytes; figures for structured types on SCI are not published. CrispORB [9], developed by Fujitsu labs, is targeted to VIA in general and Synfinity-0 networks in particular. Its latency is noticeably better, up to 25 % than with standard IIOP.

OmniORB2 was developed in 1998. In the next version, OmniORB3, there is only TCP support. Support of high-speed networks did not seem promising and thus had been discontinued. CrispORB is interesting but restricted to VIA. TAO is targeted to predictability and quality of service. As far as we know it has not been deployed on high speed networks.

3.2 CORBA Performance Analysis

This section analyzes the performance of available CORBA implementations so as to understand where are the overheads. Copy limitations are also validated thanks to two prototypes on top of high speed networks.

We will first analyze a remote method invocation. The steps are: a) build and send a header to notify to the remote object it has to invoke a method. This is the invocation latency, t_1. b) marshal and send the *in* parameters of the method. This is described by the bandwidth, B_{in}. c) execute the remote method with duration t_{exec}. d) marshal and send the *out* parameters, with bandwidth B_{out}. e) notify to the caller that the remote invocation is finished. The termination notification is t_2.

Our measurements are $RTT = t_1 + t_2$ (round trip time), which is the time needed for the remote invocation of an empty method, and the bandwidth B. If a method takes parameters of size S bytes and is invoked in time T, then $T = RTT + \frac{S}{B}$, and then $B = \frac{T - RTT}{S}$.

Coupled codes of numerical simulation handle huge amounts of data. The bandwidth is thus an important performance factor. It is determined by two factors: the marshaling/demarshaling speed and the network bandwidth.

Marshaling/demarshaling is the action of encoding/decoding data into an interoperable format called CDR – Common Data Representation – in order to put it into GIOP requests. Some ORBs use a straightforward approach; they assemble and disassemble requests by making an explicit copy of all the parameters. Some ORBs use a zero-copy strategy. Depending on the memory bandwidth/network bandwidth ratio, the copy can be a negligible or a very cost effective operation. The overall bandwidth B is given by the formula:

$$B = \frac{1}{\frac{1}{B_{marshal}} + \frac{1}{B_{net}} + \frac{1}{B_{demarshal}}}$$

We realized a minimal, not-fully functional porting of two open-source CORBA implementation on top of Madeleine : MICO [18] and OmniORB3 [1]. We were then able to measure the performance we could get from a complete implementation. We ran benchmarks on our dual-Pentium II 450 based PC cluster with Ethernet-100, SCI, and Myrinet network adapters.

Table 1 shows the peak bandwidth analysis of MICO. On high-speed networks such as SCI and Myrinet, $\frac{1}{B_{marshal}}$ and $\frac{1}{B_{demarshal}}$ become dominant. Thus, be-

Table 1. MICO's peak bandwidth analysis in MB/s

network	$B_{marshal}$	$B_{demarshal}$	B_{net}	B	$B_{measured}$	$B_{measured}/B_{net}$
Ethernet-100	129	80	12	9.6	9.4	78%
SCI	129	80	86	31	27.7	32%
Myrinet	113	72	99	30.4	26.2	26%

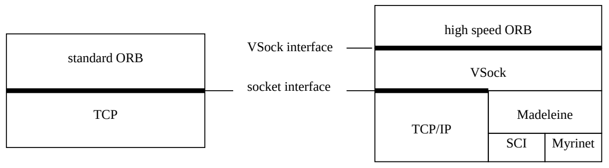

Fig. 1. Porting scheme overview

cause of the high overhead introduced by copies, the overall bandwidth B is only about 30% of the network bandwidth B_{net}.

OmniORB3 does not always copy on marshaling/demarshaling. It implements a "zero-copy" transfer mode and pre-allocated buffers as often as possible. Thanks to this strategy, it can achieve a higher bandwidth, even if theoretically the more complex marshaling methods cause a higher latency. Our OmniORB/Madeleine reaches 86 MB/s on SCI and 91 MB/s on Myrinet. $B_{marshall}$ and $B_{demarshall}$ do not make sense in zero-copy strategy. Overall performance results are given in Section 3.4.

3.3 Porting OmniORB on Top of Madeleine

The previous section has shown that OmniORB3 is well suited for high performance thanks to its efficient marshaling/demarshaling strategy. In this section, we present a complete port of OmniORB3 on top of Madeleine, our approach is to modify OmniORB as little as possible to be able to follow its next versions with virtually no supplementary work. We only modified OmniORB transport and threads layer. Porting the OmniORB thread system on top of Marcel is straightforward since Marcel implements the subset of PosixThreads API that OmniORB needs. For the transport layer, our approach relies on the concept of *virtual socket*, or `VSock`, as shown on Figure 1. `VSock` implements a subset of the standard socket functions on top of Madeleine, for achieving high-performance (ie. only datagram, no streaming). It performs zero-copy datagram transfer with a socket-like connection handshake mechanism using standard IP addresses. Then, porting OmniORB on top of `VSock` is straightforward. We realized a fully-functional porting of OmniORB on top of `VSock`.

Interoperability Interoperability is one of our main concerns. We need our high-speed ORB to be interoperable with other "non-Madeleine aware" ORBs. This implies the `VSock` module to be transparent in three ways:

Protocol auto-selection. The CORBA application built on top of the ORB is a standard application. It does not have to know that there are several underlying network protocols. Thus, `VSock` should automatically select the adequate protocol to use according to the available hardware.

IIOP pass-thru. For interoperability issues, our ORB must be able to communicate with the outside world using the CORBA standard IIOP protocol. `VSock` should determine itself whether an object may be reached using Madeleine or if it should revert to standard TCP.

Address mapping. Since we do not modify much the ORB, and for compatibility reasons, contact strings are always IP addresses. `VSock` translates, when needed, IP addresses into Madeleine logical node number using a reverse address resolution table.

`VSock`'s strategy for managing both TCP and Madeleine is the following: when a client tries to connect to a server, it resolves the provided IP address into a Madeleine address. If it fails, then the object is outside the cluster, and it reverts to standard IIOP/TCP. If it succeeds, then it asks the server if this particular object is handled by Madeleine – a machine being in a `VSock` cluster does not imply that all its CORBA servants are `VSock`-enabled! This is performed by comparing the corresponding TCP port numbers.

Dynamicity. The network topology of CORBA is dynamic, client-server oriented. The network topology of Madeleine (like most communication library for SAN) is static. A solution is to use a unique bootstrap binary that is started on each node. Thus, it satisfies the "SPMD" approach of the communication library. Then, this bootstrap process dynamically loads the actual application binaries stored into dynamically loadable libraries. Thanks to this mechanism, different binaries can dynamically be loaded into the different nodes of a grid system.

3.4 Performance

The bandwidth of our high-performance CORBA implementation is shown on Figure 2. We ran our benchmark on "old" dual-Pentium II 450 machines, with Ethernet-100, SCI and Myrinet 1, and "up to date" dual-Pentium III 1GHz with Myrinet-2000. The benchmark consists in a remote invocation of a method which takes an *inout* parameter of variable size. The peak bandwidth is 86 MB/s on SCI, 101 MB/s on Myrinet 1 (not shown on figure), and 240 MB/s on Myrinet-2000. This performance is very good. We reach 99 % of the maximum achievable bandwidth with Madeleine.

Figure 2 shows a comparison of the bandwidth of MPI/Madeleine [3] and our OmniORB/Madeleine. For small messages, CORBA is a little slower than MPI, because of the software overhead introduced by the ORB. For larger messages, our CORBA implementation outperforms MPI on SCI and has the same performance than MPI on Myrinet. The overall performance of CORBA is thus comparable to MPI. This validates our approach of using both MPI and CORBA for a better structuration of the applications without performance loss.

On the "old" machines (Pentium II 450, SCI or Myrinet 1), the latency of our CORBA is around 55 μs. It is a good point when compared to the 160 μs latency of the ORB over TCP/Ethernet-100. However, MPI/Madeleine latency is 23 μs. On the "up to date" machines (Pentium III 1GHz, Myrinet-2000), the latency

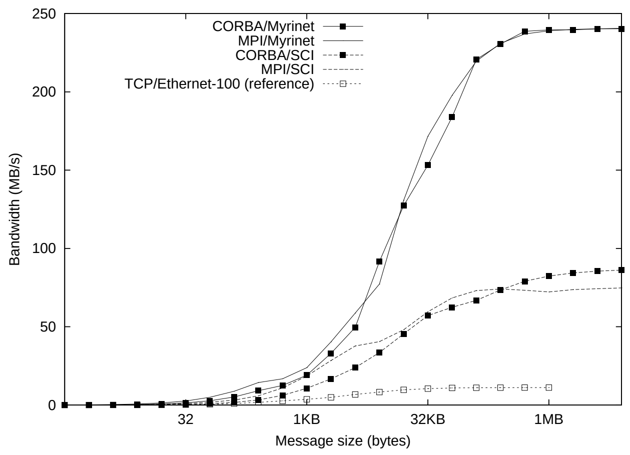

Fig. 2. Bandwidth (in MB/s) of OmniORB and MPICH over SCI and Myrinet-2000

of CORBA is 20 μs where MPI gets 11 μs. This high-performance CORBA uses the GIOP protocol. GIOP is very time-consuming and is not needed inside a homogeneous part of a grid system. CORBA enables us to write other protocols than GIOP, called ESIOP. Thus, to lower the latency, it is possible to write a high performance network ESIOP. However, figures show that the increase of the CPU power narrows the latency gap between CORBA and MPI.

4 Concurrent Support of CORBA and MPI

4.1 Problem Overview

This section exposes the problem of concurrently supporting both CORBA and MPI interface active. An ORB or a MPI implementation, and in a more general way every networkable middleware, takes an exclusive access on the resources. For example, the ORB uses the Myrinet network with the BIP protocol and Marcel threads. It is correct as a standalone package. Assume that MPI uses the Myrinet network with BIP, and Posix threads; it is fine as a standalone package. But, if this ORB and this MPI are used together, several problems arise:

- cluster-oriented protocols (BIP on Myrinet, SISCI on SCI) are most of the time single-user. They cannot be used concurrently by several packages that are not aware of the others;
- an application runs into trouble when mixing several kinds of threads;
- if ever we are lucky enough and there is no resource conflict, there is probably a more efficient way than putting side by side pieces of software that do not see each other and that act in an “egoistic” fashion.

We aim at making it work in a *coherent rather than competitive* way. The main points are: network multiplexing and common thread management.

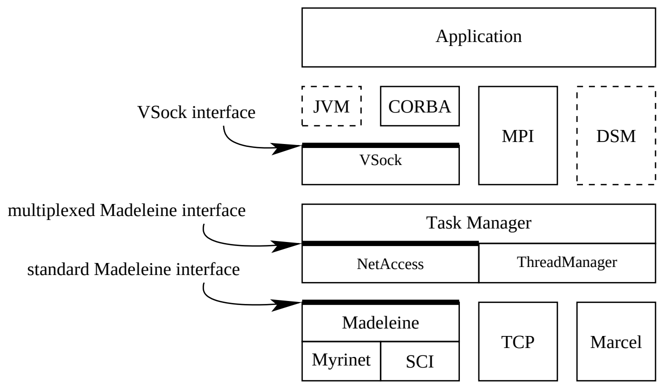

Fig. 3. Concurrent access to resources through a Task Manager

4.2 Network Access

There is a need for a multiplexing method. If both the ORB and MPI access the network resources without being aware they do not have exclusive access, there will be conflicts. The ORB is our `VSock`-based OmniORB; as an MPI implementation, we choose the Madeleine-based port of MPICH [3] for its good overall performance and its portability. We manage two level of multiplexing as shown on Figure 3:

– low-level multiplexing, provided by the Task Manager on top of Madeleine. It enables several modules to use Madeleine native communications;
– high-level multiplexing, provided by `VSock`. It enables several modules to use virtual socket on top of Madeleine. `VSock` itself is a module that uses low-level multiplexed Madeleine communications.

Multiplexing on top of Madeleine is performed by adding a tag into headers. We centralize the global operations such as initialization and channel management. Very few changes have to be done to existing Madeleine-based modules to obtain multiplexed Madeleine modules. As for the `VSock` porting, this can be automated with a script acting on the source code.

4.3 Threads Management

When it comes to multi-threading, every standalone package has its own library, compatibility layer, and policy. When we put side by side such packages that are not aware of each other, some problems arise: at best, the efficiency is sub-optimal; at worst, incompatibilities appear at run-time or compile-time.

We propose that the Task Manager centralizes threads execution, and in particular threads dedicated to communications. We chose Marcel threads for their efficiency and good integration with Madeleine. Then, we are able to have a unified thread policy:

– If every module (ORB, MPI) has its own communication thread, resources are wasted. Latency is increased because of the thread scheduler overhead. The Task Manager runs a *polling thread*. Each module may register its polling action that will be called by the Task Manager. There are, for example, Madeleine callbacks for Madeleine multiplexing.
– Since the Task Manager knows every polling function, it is able to decide on a coherent polling policy. It interleaves the several actions in a coherent way. It adapts the polling frequency to the network performance. For example, control channels are polled less often so that they do not interfere with time-critical data channels. TCP sockets are polled less often than SCI or Myrinet channels since their polling is more time-consuming.

Performance The result of this coherent concurrent support of both CORBA and MPI has a good overall performance. Every level of interface (multiplexed Madeleine, `VSock`) is zero-copy, thus the bandwidth remains unchanged at any level of multiplexing. Thanks to header piggy-backing, multiplexing does not increase latency. We are able to keep CORBA and MPI at the same performance level as when they were standalone packages as described in Section 3.4.

5 Padico

Padico is our research platform for parallel and distributed computing. In particular, it targets code coupling applications based on the concept of parallel CORBA objects [6]. The runtime environment is called Padico Task Manager, shortened in PadicoTM. The role of PadicoTM is to provide a high performance infrastructure to *plug in* middlewares like CORBA, MPI, JVM, DSM, etc. It offers a framework that deals with communication and threads issues, allowing different middlewares to efficiently share the same process. Its strength is to offer the same interface to very different networks.

The design of Padico, derived from the software component technology, is very modular. Every module is represented as a component: a description file is attached to the binary (in a dynamically loadable library form) that describes it. PadicoTM implements the techniques described in Section 4, namely network multiplexing, provided by the Padico NetAccess module and thread management, provided by the Padico ThreadManager module. Padico NetAccess and Padico ThreadManager, built on top of Madeleine and Marcel, are the core of PadicoTM. Then, services are plugged in PadicoTM core. These services are: a) the virtual socket module `VSock`, used by CORBA. It may be used by several other modules at the same time; b) the CORBA module, based on OmniORB3, on top of `VSock` as described in Section 3; c) the MPI module, derived from MPICH/Madeleine [3]; d) a basic CORBA gatekeeper that allows the user to dynamically load modules upon CORBA requests.

Currently, we have a functional prototype with all these modules available. Its performance is reported in Section 3.4. Padico is just in its beginning phase. Several important issues like security, deployment and fault tolerance are not yet addressed.

6 Conclusion

The Grid offers an heterogeneous environment, in particular with respect to communication protocols. At the programming level, it is not realistic to consider all links as similar as they are not. A better solution seems to keep the structure of the applications to have some knowledge about the performance requirement of the links. For example, a parallel MPI code expects low latency and high bandwidth communications while a code coupling communication do not expect to be so efficient. Targeting code coupling applications, our choice is not to constraint the communication paradigm used inside parallel code and to use CORBA for coupling communications. As coupling communications can be mapped on high performance networks, it is important that CORBA could efficiently exploit them. Also, as applications may simultaneously use for example MPI for its internal communications and CORBA for coupling, it is also mandatory that both middlewares efficiently co-habit. This paper shows that both requirements can be fulfilled.

First, this paper has shown that CORBA can be as efficient on high performance network as MPI. We measure 240 MB/s bandwidth for CORBA on top of Myrinet-2000. This is the same bandwidth than MPI. The latency is less than twice the MPI latency. This is mainly due to GIOP related overhead. Second this paper shows that different middlewares can efficiently co-habit in a high performance environment. This paper has given some insight on the different interactions, mainly related to network access and thread issues. This co-habitation has been obtained without loss of efficiency neither for CORBA nor for MPI. These contributions have been integrated into an operational research platform, called Padico.

These works have several perspectives. The first direction is related to CORBA. In order to reduce the latency of CORBA requests, customized CORBA protocol based on ESIOP should be studied. This can be directly be done by porting TAO on top of PadicoTM. The other direction is to plug other middlewares on top of PadicoTM as applications may want other middlewares than MPI. This also allows us to evaluate whether the concepts handled by the PadicoTM layer are adequate to DSM middleware or to Java Virtual Machine middleware. Last, we plan to use Padico as a experimental platform for parallel CORBA objects.

Acknowledgments We would like to thank the PM2 developer team for their efficient support of Madeleine and Marcel.

References

1. AT&T Laboratories Cambridge. OmniORB Home Page. http://www.omniorb.org.
2. O. Aumage, L. Bougé, J.-F. Méhaut, and R. Namyst. Madeleine II: A portable and efficient communication library for high-performance cluster computing. *Parallel Computing*, March 2001. To appear.

3. 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)*, San Francisco, April 2001. IEEE. To appear.
4. L. Bougé, J.-F. Méhaut, and R. Namyst. Efficient communications in multithreaded runtime systems. In *Parallel and Distributed Processing. Proc. 3rd Workshop on Runtime Systems for Parallel Programming (RTSPP '99)*, volume 1586 of *Lect. Notes in Comp. Science*, pages 468–482, San Juan, Puerto Rico, April 1999. In conj. with IPPS/SPDP 1999. IEEE TCPP and ACM SIGARCH, Springer-Verlag.
5. 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.
6. 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 Intl. Euro-Par'01 conf.*, Manchester, UK, 2001. To appear.
7. The OpenMP Forum. OpenMP fortran application program interface, version 1.1, November 1999. available from www.openmp.org.
8. 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.
9. 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.
10. K. Keahey and D. Gannon. PARDIS: A Parallel Approach to CORBA. In *Supercomputing'97*. ACM/IEEE, November 1997.
11. F. Kuhns, D. Schmidt, and D. Levine. The design and performance of a real-time I/O subsystem. In *Proceedings of the 5th IEEE Real-Time Technology and Applicati ons Symposium (RTAS99)*, Vancouver, Canada, June 1999.
12. Sai-Lai Lo and S. Pope. The implementation of a high performance ORB over multiple network transports. Technical report, Olivetti & Oracle Laboratory, Cambridge, March 1998.
13. Mercury Computer Systems, Inc. and Objective Interface Systems, Inc. and MPI Software Technology, Inc. and Los Alamos National Laboratory. Data Parallel CORBA - Initial Submission, August 2000.
14. Object Management Group. Request For Proposal: Data Parallel Application Support for CORBA, March 2000.
15. S. Pope and Sai-Lai Lo. The implementation of a native ATM transport for a high performance ORB. Technical report, Olivetti & Oracle Laboratory, Cambridge, June 1998.
16. T. Priol and C. René. COBRA: A CORBA-compliant Programming Environment for High-Performance Computing. In *Euro-Par'98*, pages 1114–1122, September 1998.
17. 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.
18. A. Puder. The MICO CORBA Compliant System. *Dr Dobb's Journal*, 23(11):44–51, November 1998.