Communication Kernel for High Speed Networks in the Parallel Environment LANDA-HSN

Thierry Monteil, Jean Marie Garcia, David Gauchard, Olivier Brun
LAAS-CNRS
7 avenue du Colonel Roche 31077 Toulouse, France
monteil@laas.fr

Abstract

Due to the development of several new technologies concerning high speed networks, one can get PC clusters for intensive computing at a low price. Parallel programming environments must take into account the heterogeneity of workstation networks into their communication kernels. New technologies must be easily integrated into such environments. The LANDA-HSN project has been designed for an efficient use of the available resources (network, processor, memory). This paper describes the mechanisms and the implementation of its communication kernel on SMP hosts and through interconnection networks. The contribution of this paper is to propose a flexible and efficient communication kernel.

1 Introduction

Due to the development of several new technologies concerning high speed networks, one can get PC clusters for intensive computing at a low price. Parallel programming environments must take into account the heterogeneity of workstation networks into their communication kernels. New technologies must be easily integrated into such environments. The LANDA-HSN project has been designed for an efficient use of the available resources (network, processor, memory). This paper describes the mechanisms and the implementation of its communication kernel on SMP hosts and through interconnection networks. The contribution of this paper is to propose a flexible and efficient communication kernel. The very interesting price / performance ratio makes clusters of workstations often dedicated to be parallel machines. Indeed, such a machine offers a very high potential power. They appear in the top 500 of the most powerful machines (there are 24 of them in june, 99) [7].
Because the networks are often slow, that sort of parallel machine is generally used for "coarse grain" algorithms. However, this handicap is partly hidden by the latest high-speed network hardware (SCI [9], ATM [21], Myrinet [5], MPC [3]). This kind of network will soon reach high performances.
The work presented in this paper deals with the conception of a new communication kernel in the LANDA [16](Local Area Network for Distributed Applications) environment to implement high speed communications on these networks.
The LANDA project was started in 1989 to build a general parallel environment on top of clusters. It was created by the OFP team at LAAS laboratory (Laboratoire d'Analyse et d'Architecture des Systèmes).
The LANDA software has a complete environment with graphical interfaces (trace, load, dynamic state) and allows to run and monitor parallel applications. The whole system is implemented on top of a set of daemons running on each workstation of the virtual machine. This organization allows a global view of the machine, particularly its load and its availability. The heterogeneity and the structure of the network and its architecture are transparent to the LANDA user. Tasks of a parallel application communicate via message passing and task localization is transparent to the user. The control provided by LANDA is close to an operating system if we consider its global view of the virtual machine.
The communication system implemented in the release 3 of LANDA, called LANDA-HSN (High Speed Network), is based on an optimal use of the computer architecture and network resources. It is based on the separation of the messages into two parts: header and data. The objective of the new communication system is to distribute and optimize the mailbox system in order to optimize communication operations.
LANDA is coupled with the Network-Analyser system which allows to analyze and predict the network and host loads. Network-Analyser is an observation tool (CPU load, processes states, input/output, global traffic and point to point communications) that gives all significant load informations and is also a decision tool that computes optimal message routing tables and task allocation strategies [12].

Network-Analyser provides a library which can be used by all systems (LANDA [16] communication kernel, PVM [20]) to get informations.
In the following sections, we focus on the communication model that we are designing in LANDA-HSN. We will first describe the virtual machine architecture defined in LANDA. Then we describe the communication schemes between tasks on SMP hosts and through interconnection networks. We conclude by giving the first results on our myrinet cluster.

2 Related work

Numbers of projects concerning PC clusters are developed throughout the world. One of the well known project is Beowulf in Goddard Space Flight Center (1994) with PCs running Linux, but many others exist: LOKI (Los Alamos National Laboratory), Hyglac (Caltech/JPL), PopC (MATRA Système et Information), NOW (University of California, Berkeley). Moreover, number of parallel programming environment were first designed for local workstation networks or parallel machines. Several programming paradigms are available: automatic parallelization (HPF [1]), direct message passing (PVM [20], MPI [4], LANDA [16]), shared memory (OpenMP [6]), writing in memory (VIA [8]). It is then natural to use these environments on clusters although they are not directly designed for them. However, clusters are often built with very different communication hardwares and protocols which implies that communication layers and user libraries have to be adapted to obtain the highest throughput. Hence, a particular communication layer for each protocol is necessary, but these layers have to be generic enough to be easily integrated in an heterogeneous environment, and flexible enough so as to avoid re-writing the whole library every time a new type of network appears. LANDA-HSN proposes to integrate such features. Other environments like MP_MPICH [13] or PM2 [17] with Madeleine [11] also do this.

3 LANDA-HSN Architecture

3.1 Virtual machine model

A virtual parallel machine may have a complex architecture (figure 1), due to the heterogeneity of the hardware: workstation, cluster, parallel computer. Interconnection networks may be also heterogeneous (topology, speed, protocol). If one wants to use such an architecture efficiently, a good knowledge of the architecture is required. LANDA takes into account three levels of network interconnection inside a parallel virtual machine :

- Physical level (hub, bus, switch) and physical connections (port number).
- Interface level which is the view that the operating systems has (network interface names).
- Network level which concerns the link between hosts and networks, or networks and networks.

This accurate description allows the LANDA kernel to manage efficiently communications. The physical level gives informations to create the routing tables, the two other levels give enough informations to be able to spread the tasks over the networks so that resources are used as efficiently as possible. Moreover, the Network level allows parallel tasks to be run on different sites.

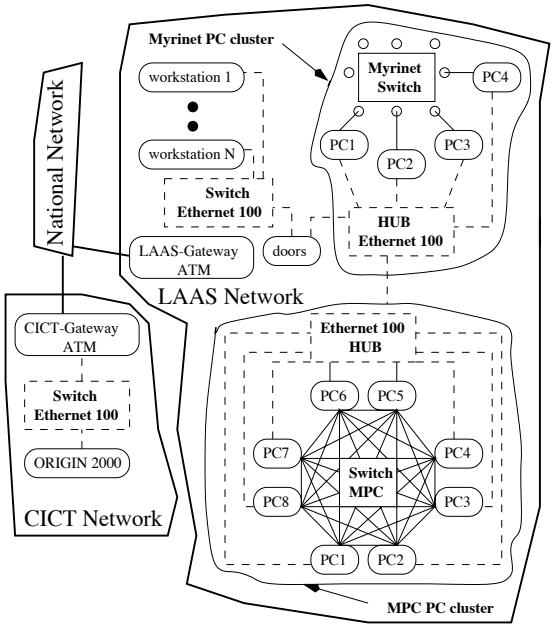

Figure 1. Architecture of the virtual parallel machine as viewed by LANDA

3.2 Communication Model

3.2.1 Dynamic Message routing

The communication model is based on the message-passing paradigm. Messages are cut into two different parts: the header and the data. Headers are always stored in the host receiver header mailbox. Data can be stored anywhere in the virtual machine, thus avoiding multiple copies of data in case of collective communications. Separation of header and data also allows to choose among several communication models according to the message storage policy.
With several memory locations, the communication system can be partially or totally distributed. In this case, several

routing policies are possible. The three basic routing policies are the following:

- Data are always sent to the receiver's host.
- Data always stay on the sender's host.
- Data are stored on a less loaded node (different from sender and destination).

These different routing policies allow a better network bandwidth utilization for remote communication depending on the load of the workstation network. The routing policies can also be used for local communication in case of local memory lack. We are studying dynamic routing strategies taking into account real time load of the whole machine [15]. The Network-Analyser system will be in charge to give real time measures to LANDA in order to change dynamically message routing tables.

3.2.2 Communication library

A high-level library, called central library (figure 2), implements all basic mechanisms and functions to manage communications, parallel programs and the virtual machine. The central library offers every functionality needed to implement the high level libraries: PVM [20], MPI [4] [14], VIA [8]. A shared memory allocation mechanism allowing zero-copy local communication is also provided by this library.

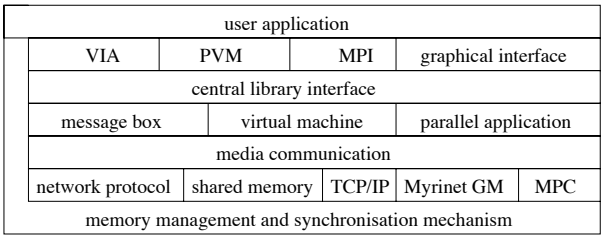

Figure 2. Landa architecture

3.2.3 Kernel communication library

High-level communication layers are built on top of the media communication layer. Thus any kind of hardware can be transparently used through it. For each communication, LANDA chooses the best hardware link according to the data priority. LANDA defines several communication types:

- kernel management communication
- user data communication
- urgent communication

The interface of the media communication layer provides different functions concerning communication links or channels. Apart from the 'open' and 'close' functions, we have blocking and non-blocking 'read' and 'write', and we are planning to use call-back functions to tell the availability of the data from one or more channels. The figure 3 shows the lower layers that the media communication layer depends on. Each one is represented by a class (in object programming language). The media communication class is a set of channel classes. The channel class offers virtual methods that are really implemented by each particular hardware's class. Thus it is easy to add and use new communication hardware.

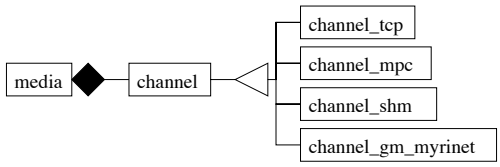

Figure 3. Communication class

4 Principle for local communication

4.1 Design

We have decided to use shared memory (SHM) and semaphores for local communications between processes on the same multiprocessor host, for the following reasons:

- We may have only one receiver per host for a particular communication medium (eg: GM/Myrinet), so the data must be able to be shared between all the processes.
- Zero-copy communication between two or more processes is possible.

So far mmap()-ed shared memory and IPC-semaphores are used. However, the software has to be independent of system mechanisms. So we have wrappers that allow to change the underlying system calls according to the architecture's capabilities - for instance thread's mutexes cannot be shared between processes on Linux, so we use IPC semaphores, whereas we can use them on Solaris to enhance performances. The figure 4 shows a 'ping-pong' performance comparison using semaphore or mutex for synchronisation under Solaris.

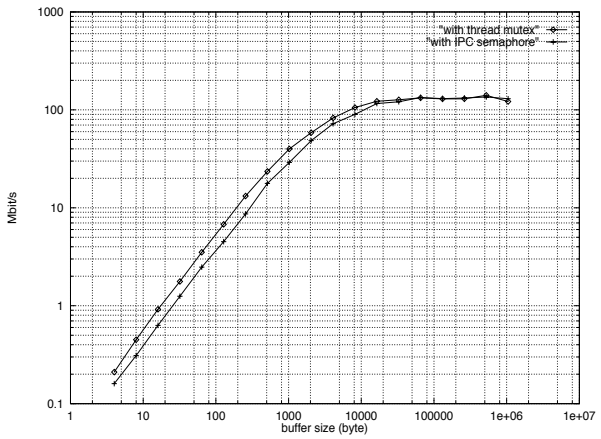

Figure 4. IPC-semaphore vs threads-Mutex

4.2 Memory management

The objective is to create a shared memory as easily usable as the local memory with some useful tools to share data between two or more processes. This means that any process can ask for a shared memory chunk of any size at any time. To be able to provide this service, we use an external memory allocator since we cannot use the system allocator for the shared memory segments we create. This allocator was made by Doug Lea and is available on the Internet [2]. This allocator is the one used by the Gnu-Libc on Linux. It has been slightly modified for LANDA to be usable in a distributed environment.

To share data between processes in the shared memory, we use what we call a buffer. A buffer is a structure that contains a pointer to the data themselves in memory, and some informations about them, particularly a reference counter. We need to know the number of references to these data from all the processes, so that the data released by a process are not freed from the memory if other processes access to them. Consequently the buffer is the only structure used by all the communication functions. However, using shared data implies that they have to be read-only by all the receiving processes.

The landa system classes for local communication are organized as shown in figure 5.

- Class 'sembase' handles semaphores creation and management
- Class 'mallocore' handles Doug Lea's malloc (malloc/free)
- Class 'shmchunk' handles allocation and management

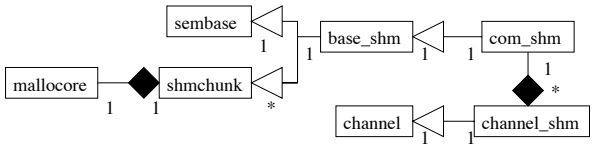

Figure 5. class organization for local communication

of shared memory segments

- Class 'base_shm' group a list of shmchunk class and the semaphores class
- Class 'comm_shm' implements the circular arrays (see in next section)

4.3 Local communication using shared memory

Each local process has a circular array used to store buffer references that come from any other local process. When a process needs to read data from the shared memory link, buffer references are read from this circular array, then stored in local memory using a different list for each different sender.
The reason why we use circular buffers to temporary save the buffer references is that this structure is very fast to manage, and we do not want the senders to spend time in managing other processes data.
When data are small enough to fit in a single communication, header and data are grouped in the same memory chunk. This allows making less communications, it will be particularly useful in inter-hosts communications. It is also a solution to keep communication latency low.
In the example on figure 6, Task 2 sends a message to Task 1. First, the message is either copied into or directly created in shared memory, then a header that describes the communication is built. T2 communication work consists only in putting a reference to those data in T1's circular array. The semaphore semantic is used to protect and synchronize the processes. T1 then reads the references in the circular array, possibly put them in its own lists if the data are not interesting for now, and then reads them from the shared memory.

5 Remote communication

5.1 Design

Communication between remote hosts is done using a daemon called "communication daemon" (CD). In general, there is one CD per user and per host. A CD must be shared by the various applications of users. It is also possible to

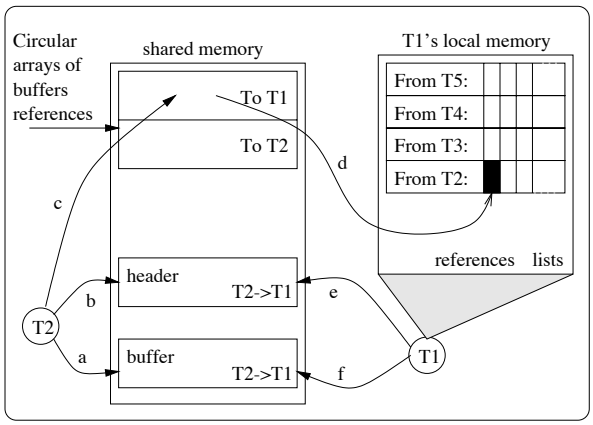

Figure 6. shared memory communication

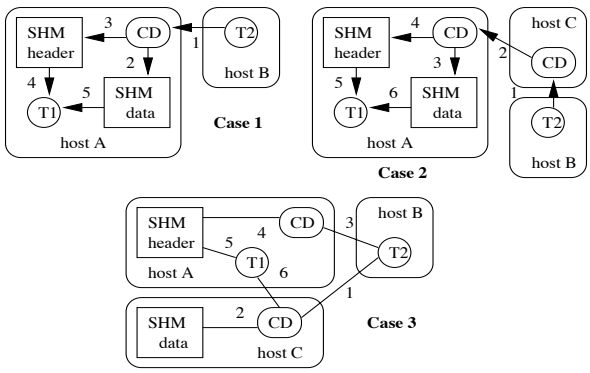

Figure 7. Communication Mechanisms

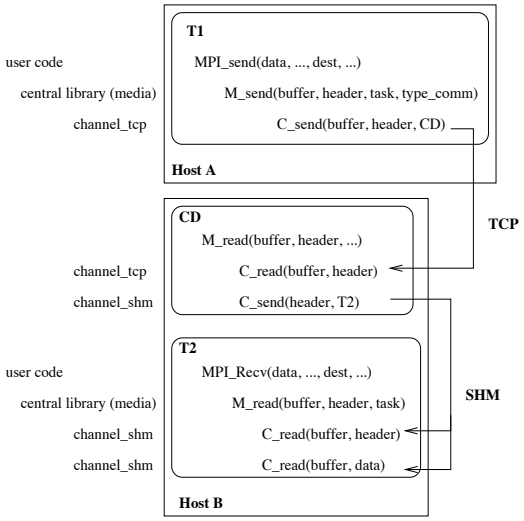

Figure 8. Communication over TCP

have only one CD for all users on a host. This provides a solution to the limitation imposed by some communication hardware (Myrinet/GM, MPC).

Sending a message is done by various request mechanism to the destination task's CD (i.e. the CD running on the same node). One can distinguish 4 different cases (figure 7):

- Header and data are stored on the destination task's host, and the latter is directly reachable.
- Header and data are stored on the destination task's host, but this host is not directly reachable. Then the network view of the virtual machine must be used to send the message from CD to CD until the destination CD.
- Header is stored on the destination task's host, but the data are stored elsewhere.
- Like the previous case, but with hosts not directly reachable.

The sender task always reaches the destination task's CD to send a message. The receiver task reads the header on its CD, and then reads data through the specified CD.

5.2 Programming structure

The communication mechanism depends on the network. For example for TCP (figure 8), the task T1 calls the MPI function MPI_send. This function calls the central library media_send() function. A buffer and a header are build. The data sent don't need to be post-processed if both sender and receivers are homogeneous, otherwise the central library extracts the characteristics of the receiver task and the type of communication. After this, it uses the class media to send the message. The media chooses the best channel for this communication and sends the message to the remote CD. The CD on the host B is waiting for messages on its media, that is to say on all the possible channels. The CD then reads the message with a read() function of the channel virtual class which is implemented by a channel_"hardware" class and put them in shared memory. A header to the local task is then given to T2 using a send() of the channel_shm class. When the task T2 asks for the message by using MPI_Recv(), it uses the central library and the media class.

5.3 Myrinet Implementation

5.3.1 Use of GM library

GM is one of the message-passing system for Myrinet networks [10] [18] [19]. The GM system includes a driver, Myrinet-interface control program, a network mapping program, and the GM API. The system provides a 560 Mbits/s bandwidth between two PII-233 with a 38us latency. GM features include:

- Concurrent, protected, user-level access to the Myrinet interface.
- Reliable, ordered delivery of messages.
- Scalability to thousands of nodes.
- Very low host-CPU utilization.
- Two priority levels: low and high

Send and receive are done using tokens. Sending a message at a given priority can only be done if the sender owns a token for that port and priority. When the message is sent, the sender free the token. As sends are non-blocking, the sender receives an event to tell it that the sent messages have really been sent. Receiving a message of a given size and a given priority can only be done if the receiver owns a token matching these conditions. A communication is done as follows: the receiver asks for a message of a given size after having reserved a buffer. The sender sends the message, and when the receiver receives it, an acknowledgment is sent to the sender.

5.3.2 Design

GM's communication mechanism needs a more complex channel class. The communication kernel must give empty buffers in DMA memory to GM before application starts. These buffers have a fixed length. This means that the receiver must know before the application starts, the size of the buffer it will receive. Yet, this is not always true, and there are problems if one needs to send larger messages. Currently, we have two solutions for that:

- segment the message into chunks of the correct size
- send a request for the size needed, then send the message

Another problem still remains for event management. In order to not slow down communications, the sender does not wait for the "receive" event. A thread has been implemented to manage these events and is in charge to retransmit the messages if necessary. This thread has a small activity since messages are nearly never lost by Myrinet.

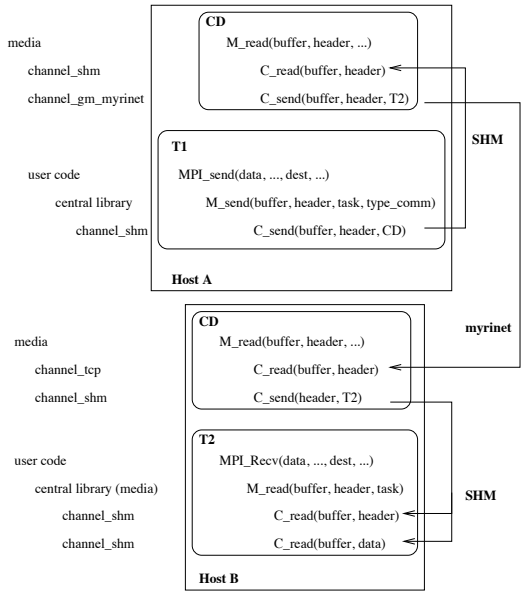

Figure 9. Communication over myrinet

Concerning Myrinet, the communication mechanism with CDs is different (figure 9) because of the limited number of GM communication ports. CDs are the Myrinet gateways for every tasks on the same host. When a task T1 sends the message, it must use its local CD. The communication between the task and the CD is done using the share memory and the channel_shm class. Then the local CD reads the header and forwards it and the data to the remote CD using Myrinet.

6 Experiments and results

6.1 Experimental Platform

The test platform (figure 1) is made of 2 clusters. There are 4 dual Intel-PII-233 using 100Mbps Ethernet TCP/IP network and GM/Myrinet (board PCI32C-Lanai4) [5] with linux installed in the first cluster. The second cluster has 2 Intel-PII-233 connected by a 100Mbps Ethernet TCP/IP network and a MPC [3] network. We will have 6 more PCs in the MPC network by the end of the year. These computers are connected to the LAAS network.

The first goal for LANDA is that the communication library makes an efficient use of the GM/Myrinet network. The management of several computer-sites will be done using an Origin-2000 parallel machine located outside our lab. The link will be TCP/IP and ATM.

Currently, the tests are done on the low level communication layers.

6.2 Local communication

The proposed tests are a simple "ping-pong" which uses the landa low layers. The tests are intended to evaluate the objects' overhead.

In the test named "2-copies test", buffers are copied from sender local memory to receiver local memory through the shared memory. In the test named "1-copy test", there is only one copy when the message is sent or received, since it is possible to work directly in shared memory. When dynamic allocation is used, buffers are allocated and freed at each data exchange using the shm-malloc/free functions.

The horizontal axis is the size of the exchanged data, the vertical axis is the communication throughput in Mbits/s. The throughput is the sum of the buffers sizes (sent and received) divided by the test duration. All the tests are run on a dual-pentiumII-233Mhz.

- In figure 10, we have made a very simple "ping-pong" test using semaphore and memcpy() only, to check the overhead of the classes.
- In figure 11, we show the efficiency improvement if only one copy is done during the data exchange, that is to say that either the sender or the receiver is directly working in the shared memory before or after communicating.

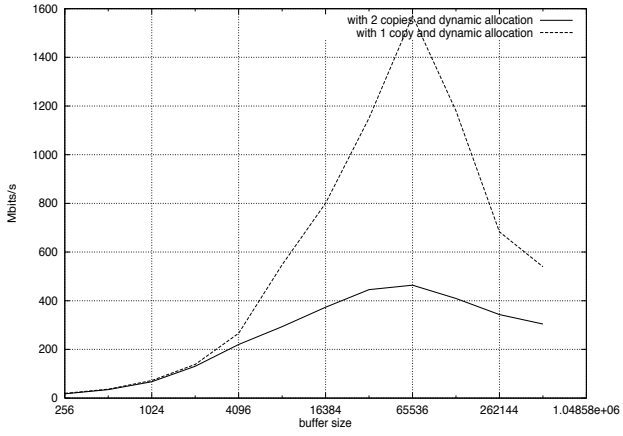

Figure 10. Library overhead

The figure 10 shows that some work has to be done to reduce the library latency. With high-speed networks, it

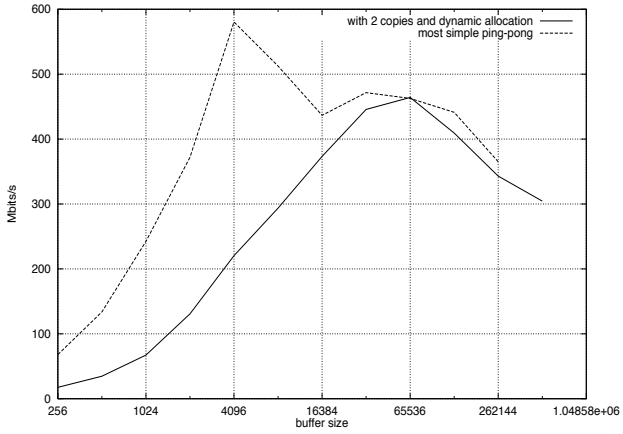

Figure 11. 1 copy vs 2 copies

is preferable to communicate with small buffers, particularly for fine grain algorithms. The figure 11 shows that it is worth using directly shared memory instead of copying data to local memory. The library allows the user to work indifferently in local or shared memory, thanks to the shared malloc/free functions. For instance, on Linux systems with much memory, it is possible to create shared memory chunks with a very large size: the maximum size of allocable shared memory can be dynamically set in the linux kernel.

6.3 Myrinet Communication

The goal of following tests is to evaluate the weight of the library itself. The test programs are on the same level as the media layer. A ping-pong is done between two processes each on a host connected to the Myrinet switch. The curves (figure 12) compare the two policies described earlier. The size of the buffers is 16KB for the reservation policy.

It is difficult to evaluate the protocol weight from these results. The best throughput is 415 Mbps. Reservation is interesting for big messages, because the overhead of segmenting and re-assembling many small messages is too high. On the contrary, segmentation is preferable for small messages, because the time needed to create a memory chunk in DMA memory is too long. The real throughput that the LANDA library will offer will be lower because the header management is not taken into account in this test.

7 Conclusion and Future Work

In this paper, we have described the new LANDA-HSN communication kernel that allows to take into account

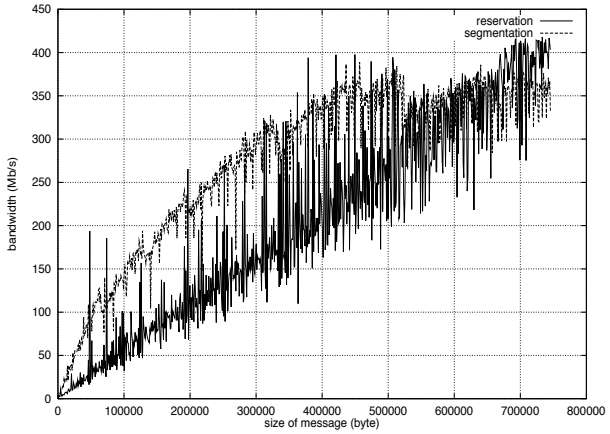

Figure 12. Communication over Myrinet

several communication protocols and the multiprocessing capabilities of SMP computers. The most interesting feature of this communication kernel is its resource management abilities: network routing, data storage selection, shared memory.
The low layers of this communication kernel are implemented. We have shown that they are efficient both for local communication on SMP nodes and for network communication on a Myrinet network. Higher layers are being written. A big part of the remaining software will come from the previous versions of LANDA.

A part of our work now focuses on the integration of new resource management algorithms:

- multi-site management.
- multi-application management (virtual machine sharing).
- load balancing by data redistribution.
- adaptive routing.

Our PC clusters will be used to run applications like Bayesian filtering, image synthesis, electromagnetism and parallel data bases.

8 Acknowledgments

This work is supported by the Midi-Pyrénées region under research project 9609558, by the defense research project DGA/DRET 95/081, and a grant from Delta Partners company.

References

[1] Hpf. *http://www.crpc.rice.edu*.
[2] malloc. *http://g.oswego.edu/dl/html/malloc.html*.
[3] Mpc. *http://www-asim.lip6.fr/mpc/*.
[4] Mpi. *http://www.mpi-forum.org*.
[5] Myrinet. *http://www.myri.com*.
[6] Openmp. *http://www.openmp.org*.
[7] top500. *http://www.top500.org*.
[8] Via. *http://www.viarch.org*.
[9] *IEEE Standard for Scalable Coherent Interface*. IEEE Std 1596-1992 I. C. Society, Inc 345 East 47th Street, New York, 1993.
[10] N. Boden, D. Cohen, R. Feldermann, A. Kulawik, C. Seitz, and W. Su. Myrinet : A gigabit per second local area network. *IEEE-Micro*, (15-1):29–36, February 1995.
[11] L. Bougé, J. F. Méhaut, and R. Namyst. Madeleine: An efficient and portable communication interface for rpc-based multithreaded environments. *Int. Conf. on Parallel Architectures and Compilation Techniques PACT'98*, pages 240–247, october 1998.
[12] J. M. Garcia, D. Gauchard, T. Monteil, and O. Brun. Process mapping given by processor and network dynamic load prediction. *European Conference on parallel computing Euro-Par'99*, pages 291–294, august 1999.
[13] W. D. Gropp and E. Lusk. User's guide for mpich, a portable implementation of mpi. *Mathematics and Computer Science Division, Argonne National Laboratory*, (ANL-96/6), 1996.
[14] M. Lauria and A. Chien. Mpi-fm: High performance mpi on workstation clusters. *Journal on Parallel and Distributed Computing*, 40(1):4–18, 1997.
[15] T. Monteil. *Etude de Nouvelles Approches pour les Communications, l'Observation et le Placement de Tâches dans l'Environnement de Programmation Parallèle LANDA*. PhD; LAAS-CNRS France, 1996.
[16] T. Monteil, J. Garcia, and P. Guyaux. Landa: une machine virtuelle parallèle. *Revue calculateurs parallèles*, 7(2):119–137, 1995.
[17] R. Namyst and J. F. Méhaut. pm^2: Parallel multithreaded machine. a computing environment for distributed architectures. *ParCo'95 (PARallel COmputing*, pages 279–285, september 1995.
[18] S. Pakin, M. Lauria, and A. Chien. High performance messaging on workstations : Illinois fast messages (fm) for myrinet. *Proc. of Supercomputing'95, San Diego, California*, December 1995.
[19] L. Prylli and B. Tourancheau. Bip: A new protocol designed for high-performance networking on myrinet. *Proc. of PC-NOW IPPS-SPDP'98, Orlando, USA*, March 1998.
[20] S. Sunderam, G. Geist, J. Dongarra, and R. Manchek. The pvm concurrent computing system: evolution, experience, and trends. *Parallel Computing*, (20):531–545, 1994.
[21] B. Weiss. *ATM*. Hermes, 1995.