Appendix C: Communications in the CeTAD Local Area Network

Interconnection networks are an essential part in parallel computer architectures. In fact, a large part of the parallel computing bibliography is dedicated to this subject matter. This is relevant from two points of view highly related to parallel applications; flexibility and performance. In addition, the higher the interconnection network's flexibility and performance, the higher will be the hardware cost, and the increase in this cost is usually rather more linear than the increase in the number of processors of the parallel machine.

In the particular case of parallel matrix multiplications analyzed in the experimentation, the interconnection network is of utmost importance. Performance indexes show that most of the running time of the matrix multiplication in parallel is spent for data transmission. Given the impact that the low communication performance has over the total performance of the parallel application, it is necessary to count with a precise way of characterizing the interconnection network performance in order to make the necessary decisions for the optimization of the whole application performance.

This Appendix is mainly dedicated to the characterization of the interconnection network performance of the CeTAD local area network. The results of a set of really simple experiments that allow evaluating the network quite precisely are developed and shown. Even though the message-passing library used is PVM (Parallel Virtual Machine), some of the performance characteristics common to most (if not *all*) message passing libraries implemented for local area networks are also presented, such as implementations of MPI (Message Passing Interface). It is important to notice from the very beginning the user-level process poitn oif view for the communication performance, since it can be really far from the values defined by (or available at) the interconection/communication hardware.

Although a quite large set of point-to-point communication performance results (between two processes running in different machines) are presented in PVM, broadcast message performance results are also presented. In fact, just as the parallel matrix multiplication algorithm has been designed, it is the broadcast message performance the one which affects the final performance of the matrix multiplication parallel processing. Eventually, the performance characteristics are given in LQT and LIDI local networks, in which similar experiments have been carried out; some comments on the most remarkable performance differences in these networks are presented, as well as the reasons for these differences.

C.1 Introduction

LThe processor interconnection network is fundamental in parallel computers. In the case of parallel computers with (or based on) shared physical memory, this interconnection network can be clearly identified not in terms of processors' interconnection among each other but in terms of processors interconnected with the memory. In other words, in these computers, removing processors interconnection network with the (*only*) memory would mean eliminating completely the possibility of running applications. On the other hand, if a processors' interconnection network is removed from a parallel computer with distributed memory, it stops being a parallel computer and turns into a set of separate computers or modules of CPU-Memory, without the capacity of cooperating for the solution of a problem. Since computer networks used for parallel computing clearly belong to the MIMD type of distributed memory, we will keep on considering the interconnection network for the data transmission among processors (or computers, directly).

As regards the processor interconnection network flexibility, the problem is not only to find a way of delivering data among processors, but also obtain the maximum of simultaneous communications. The typical examples in this sense are focused on the capacity or not of communicating at the same time all the possible processor pairs, or delivering information from one processor to the remaining in a single step (or in a number of steps independent of the number of interconnected processors).

The flexibility of an interconnection network will define the easiness (or not) of user applications to solve a communication among their processes. The fact that each processor will be in charge of running one or more processes to be communicated to other processes assigned to other processor/s should never be overlooked.

The view of the interconnection network performance is directly related to the data transfer time among processors of a parallel computer. This performance view is not necessary disjointed from that of flexibility. In fact, the greater the number of simultaneous data transfers among pairs of processors, the greater will also be the capability or quantity of data transferable by an interconnection network per time unit.

Even though this idea of *bandwidth* (transfer rate) or data quantity per time unit is important, another relevant performance index is the minimum communication time between two processors, or startup, or also called communication latency among processors.

In terms of costs, there is an invariant relation through the different possibilities of interconnection networks: the higher the flexibility and/or performance of the interconnection network, the higher will also be the cost. The cost increase varies according to the interconnection network used, but in many of the cases increasing the number of processors of the parallel computer implies a more than linear increase of the processor network cost. In the particular case of the installed computer networks, the cost is zero (in general, worthless), since they are already interconnected.

The main drawback of computer interconnection networks in terms of performance is that they were not designed for parallel computing. In this sense, the local network performance is placed several orders of magnitude below *traditional* parallel computers interconnection networks. That is why it is really important to evaluate their performance from the point of view of user processes that make up the parallel application.

POn the other hand, the interconnection network performance has a direct relation to the performance and granularity of parallel applications that can be run over a computer. Every communication time tends to degrade the total running time of a parallel application, unless we count with, and use to the utmost, the capacity of overlapping computing time with communications. From the granularity point of view, if the communication time to obtain a result in processor P_1 is equal to or greater than the computing time necessary to compute it, then the most reasonable is to obtain it locally (in P_1), saving time and / or complexity of the application.

C.2 Ethernet Networks

In the particular context of installed computer networks, local area networks, or LAN, the most used interconnection network is that defined by the standard protocol 802.3. This standard is also known as 10 Mb/s Ethernet network, due to its 10^6 bits per second transmission capability. The characteristics of this interconnection network are very well defined and known, in terms of hardware and of the characteristics of its flexibility and performance.

In addition, most of the 10 Mb/s Ethernet network characteristics are similar to those of the 100 Mb/s Ethernet network, in which only the parameters/indexes referring to performance or communication capacity are changed. This similarity is exemplified in, and also taken by, many of the communication hardware companies in charge of commercializing NICs (network interfase cards) which count with both communication capacities and referred to as of 10/100 Mb/s.

Fig. C.1 schematically shows the basic logic way in which workstations are connected in a local network using Ethernet. It can be easily noticed that it is of bus type, where the main characteristics of each data transfer are the following:

- There are no priorities nor the medium access time is predictable.
- It has a single sender.
- It occupies the only communication channel.
- It can have multiple receivers.
- The medium access mode is CSMA/CD (Carrier Sense, Multiple Access / Collision Detect).

The first two characteristics clearly imply that there should not be more than one simultaneous data transfer because, in fact, there is only one communication channel shared by all computers. The last mentioned characteristic makes the implementation of the *broadcast* and/or *multicast* type communications really natural, where a message is sent from a computer and received by the remaining or by a subset of the remaining computers of the network, respectively.

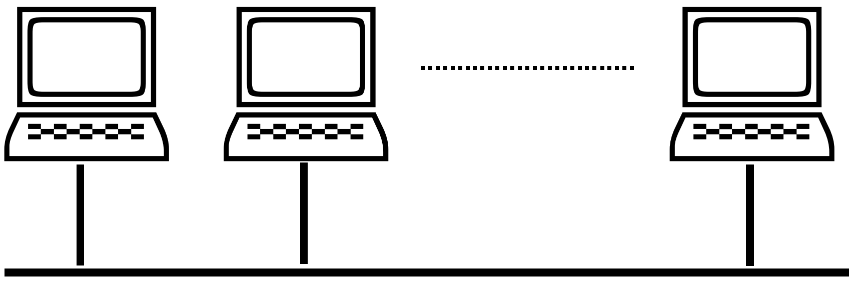

Figure C.1: Ethernet Network.

In most of the installations, the initially adopted communication hardware has been based on coaxial cable, with which the physical topology is equal to the logical topology of Fig. C.1.

Gradually, the wiring rules used in most of the installations have been changed into the use of twisted pair cable with hubs (basically, communication concentrators and repeaters) and with communication switches, which count with the facility of isolating point-to-point communications. This isolation in switches is produced when the hardware detects a point-to-point data transmission between two of the computers interconnected by a switch. Also, several combinations of hubs and switches are possible, with the objective of keeping certain performance as the data traffic increases, and also reducing the cost by avoiding the massive use of communication switches.

These communication networks are important not only due to the number of installations currently working, but also due to the fact that they are clearly less expensive than the rest of the commercialized alternatives. They are less expensive in terms of the necessary hardware (cards, connectors, and wiring) and also in terms of installation: from workforce (technicians) to hardware acknowledgement and hardware starting up on the part of the operating system. All of this necessarily reduces installation and maintenance costs of Ethernet networks.

Cost reduction in relation to the other computer interconnection alternatives implies certain guideline in the Ethernet network maintenance as well as in the installation of new networks with this hardware. We should take into account that the cost may include aspects such as: network cards in each computer, wiring (it may include hubs and/or switches in the case of Ethernet networks), installation, maintenance and technical trained personal.

C.3 Performance Evaluation

Communication time (used in general to characterize the performance of an interconnection network) between two processors is generally characterized with [7] [11]

$$t(n) = \alpha + \beta n \qquad \text{C.1}$$

where

- n is the transferred data unit to be measured (bit, byte, a simple precision floating point number representation, etc.).
- α is the time necessary to establish a communication between the two processors, which is usually called communication latency time. It basically consists of the minimum time that each communication between two processors will spend independently of the quantity of transferred data. Normally, it is given by the communication hardware and can be estimated with the time used to transfer a data unit or, when possible, a message without data.
- β is the inverse value of the asymptotic bandwidth or data transfer rate of the interconection network, i.e. $1/\beta$ is the asymptotic bandwidth. Normally, the communication network data transfer rate is given by the amount of data (bits, bytes, etc.), per time unit that can be transferred between two processors. Generally, it has a minimum limit given by the communication hardware plus the time used up by processes and/or function of the communication hardware.

$$t(n) = \alpha + \beta n \qquad \text{C.1}$$

where

Even though the communication hardware, or processor interconnection network, has well defined values for parameters α and β, the user process usually obtains worse values from the point of view of the processor interconnection network performance than those given by the hardware. Both the latency time and the time necessary to transmit each data unit are affected (and, thus, they increase) when all the communication layers necessary to make a user process message running over a processor reach another user process running in other processor take part in the transference.

It is also really difficult to make an *a priori* estimate of the overload that communication libraries - with the available users (processes), the operating system interface with the user and / or with the previously mentioned libraries, and communication protocols - impose on communications that are carried out physically over the communication medium being used. For this reason, experimental methods for measuring α and β real parameters, which user applications will find with respect to the parallel computer interconnection network, are quite often used.

EIn the specific context of computer networks, and with the possibility of heterogeneous hardware, this overload becomes more important when the processor interconnection network performance is to be evaluated. In the case of traditional parallel computers, the computation of real α and β is often easier and with more final values (what user parallel application processes obtain) closer to those of the hardware because, from the communication hardware to the interface used by user applications, everything is oriented to making parallel computation.

It is really difficult to precisely quantify the relation of local networks with traditional parallel computer interconnection networks in terms of the mentioned performance indexes. What is generally accepted is that the worst relation is given in relation to the initialization time of messages to be communicated between two processes. Moreover, in the context of heterogeneous local networks, communication initialization time usually depends on the computers used because the operating system call times are involved as well as their further overload in terms of the used protocol (or *protocol stack*) maintenance and management. At a level closer to hardware, the times of

- memory access,
- DMA (Direct Memory Access) channel initialization-use, if used, and
- the related interruptions management and / or interface with the network card of each computer to be communicated

are also involved.

Summing up, local network performance is lower than that of the interconnection processor networks of a traditional parallel machine from several points of view:

- Latency and bandwidth.
- Overlapping capacity.
- *Latency heterogeneity* depending on the machines heterogeneity.

C.4 Evaluation with the Ping-Pong Method

In the context of parallel computers with MIMD-base architecture, the experimental method for assessing the performance of the processors' interconnection network, or more specifically the real values of parameters and , has been that of *ping-pong* messages. The method is really simple in itself, since, in order to evaluate the communication time between two processors P_1 and P_2, the steps described in Fig. C.2 are to be followed:

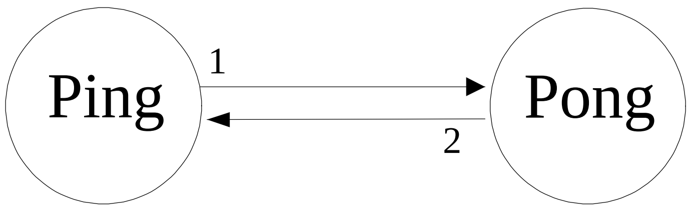

Figure C.2: *Pingpong* Processes.

1. Send a message from processor P1 to processor P2.
2. Send the received message in processor P2 to processor P1, once again
3. In processor P1, the total time used for the communication of both messages is known and, thus, this time is divided by two, achieving the message communication time in one of the directions.

One of the most attractive characteristics of this method, apart from its simplicity, consists in that it does not need the synchronization of none of the processors involved in the data transference so as to obtain a reliable data communication time. Otherwise, we should know the sending time of the message from processor P_1 to processor P_2, the reception time of the message to processor P_2, and processors should be synchronized with respect to a common time reference.

Another advantage of this method consists in the independency of the means by which messages are communicated between processes. The time spent in sending and receiving the same message in P_1 (communicating with P_1) can be taken independently of the way the message is transmitted. In fact, it is possible to analyze different available communication alternatives between processors in order to choose the most convenient.

On the other hand, one of the constraints as regards reliability of this method lies in the fact that the communication time between processors must not vary according to the *direction* of the communication, i.e. there exists certain *symmetry* of communications between processors. In the previous listing, it is assumed that the time used to send a message from P_1 to P_2 is the same as that of sending the same message from P_2 to P_1. Anyway, this situation is, without doubts, what usually happens in the fields of interconnection networks in general.

Another simplification of the *pingpong* method consists in masking, or better, disregarding the possibility of making the transmission and reception of data simultaneously (*full duplex* communication), or the possibility of overlapping computation with communication that can be used up by applications. In both cases, i.e. with the available hardware for making one or both things, having the point-to-point communications performance it is possible to reach the communication performance values that applications may obtain.

In the specific case of computer networks, the user normally does not usually have too much control (or no control) over the utilization or not of the hardware facilities. In consequence, the performance values obtained by the pingpong method will be the closest to what user (parallel) applications can obtain.

C.5 Different Ways of Message Transmission with PVM

Since it is necessary to characterize the communication time between processes of a parallel program, which are communicated using the communication routines provided by PVM, it is necessary to explore all the possible alternatives in terms of the attainable performance with these routines for user applications.

With the pingpong method previously explained, we can evaluate quite fast which the attainable performance in communications is between processes assigned to different processors of the (*virtual*) parallel machine. In addition, since we count with the values related to the performance of the communication network (α and β) of the communication hardware, we can certainly know the overload time degree, since PVM is used to communicate tasks.

When performance of communications taking into account not only the hardware but also the communication software (basically, the involved operating system processes plus PVM communication routines) has to be evaluated, it is necessary to explore the different communication alternatives to transfer data between the parallel application processes.

In general, PVM counts with two levels of flexibility when data are to be transferred among processes: data codification and “routing” (in PVM terminology) of messages. Data codification is related to the representation of the information made in each processor (computer) and the message routing is related to the way data are transferred between parallel application processes using the physical communication network and PVM communication routines/processes. In the following subsections, we present the details of the alternatives for the encoding and routing in PVM. In the particular case of the codification, we will also describe an alternative to the classical ways used both in PVM and in MPI, which will be called *Direct Translation of Data Representation.*

It is necessary to make clear that, even though this description is typical of PVM, both in MPI and in any other library used to parallel computing in computer networks, it will be necessary to define both the way in which the different data representations are matched and the way in which data are transferred over a computer interconnection network.

C.5.1 Message Data Coding in PVM

Data codification should be basically chosen from what in PVM is called:

a) *PvmDataDefault*: it is used when the communicated processes are assigned to processors with different architectures or when the application does not have any knowledge of the processors (computers) architecture over which it is run. Data to be transferred between processes are codified in XDR format before being sent; then, they are copied to a memory area from which PVM routines will send (buffer) information and then they will be decoded (from the XDR format) when they are received, before being used by the receptor process.

b) *PvmDataRaw*: it is used when the architecture of the parallel machine is homogeneous, be it a computer network or a multiprocessor. Data transferred between processes are not codified at all, they are only copied to the PVM buffers before making the delivery via the communication networks.

c) *PvmDataInPlace*: this alternative is similar to *PvmDataRaw*, but without a copy of user’s data into buffers. There is no data codification, nor extra memory costs due to data communication, nor data copying time; however, the user must assure that data will not be changed from the call of the sending routine until data are efficiently sent to the target process.

Fig. C.3 shows the three types of encoding and their relation to the codification and memory used for sending data from process P_1 assigned to a processor and process P_2 assigned to other. When the a) *PvmDataDefault* encoding is used, data are copied from the user’s process data area P_1 to the area of PVM buffers and codified in format XDR [10] in step 1. Then, these codified data are sent to the target processor (which the receptor process P_2 is assigned to), using the interconnection network, in which they are received over another area of PVM buffers. In step 2, data located in PVM buffers are decoded and copied to the data area of the receptor processor P_2.

When the b) *PvmDataRaw* encoding is used, the process is the same, with the exception of not using any type of encoding for the data that are being sent. The bytes sequence of the data area of process P_1 reaches process P_2. With this method we save buffers memory used for the XDR codification, and we also avoid using the CPU involved by such codification; however, the copies and buffers are used in the same way as in the case of using the PvmDataDefault codification.

When the c) *PvmDataInPlace* encoding is used, we not only avoid what is implied in the XDR encoding (processing and associated memory) but also all the memory necessary to store the message in the origin processor of the communication. In the case of Fig. C.3, data are sent in the first step from the user process P_1, which sends the message to the processor which the receptor process P_2 is assigned to, where it is stored in PVM buffers. In other words, output PVM buffers are no longer necessary for the message.

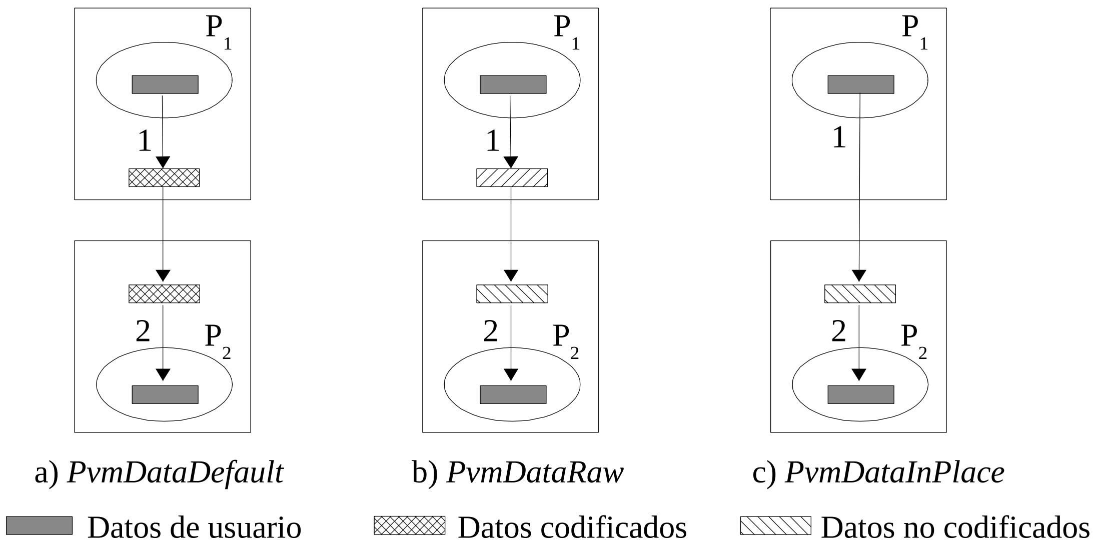

Figure C.3: Codification Alternatives in PVM.

In the examples of the performance evaluation of communications included in the PVM distribution, the way in which data are codified is *PvmDataRaw*, with a comment indicating that, in case there is heterogeneity in the parallel machine, we should change for the *PvmDataDefault* encoding. In general, in computer networks, there is no option other than codifying data (with the addition of the copies to PVM buffers) with *PvmDataDefault* so that data do not loose their meaning when they are delivered to other process executed over other processor (computer). When the used computer's heterogeneity is accepted for parallel computing, it is not possible to assume that the data representation in all computers is homogeneous.

From the performance point of view, the most appropriate codification method is the *PvmDataInPlace*. Flexibility loss due to the restriction as regards the impossibility of modifying the sent data does not seem to entail a problem in the particular case of matrices multiplication. Sent data (sub matrices of matrix B, in the operation A=BxC) are read-only for all the processes that use them. All the same, the data representation heterogeneity problem persists.

All numerical applications depend, as regards data representation, on a quite reduced quantity of data types which are generally predefined by the languages used (generally C or FORTRAN). In general, two basic types can be identified, from which data structures with which we operate in applications (mostly vectors and matrices) are defined: single precision and double precision floating point. For instance, the representation of complex numbers, also necessary for a wide range of numerical applications, is defined in function of a pair of single or double precision floating point numbers, according to the specific application.

From the three types of encoding already explained, the general strategy used for transferring data from a computer to another schematically is:

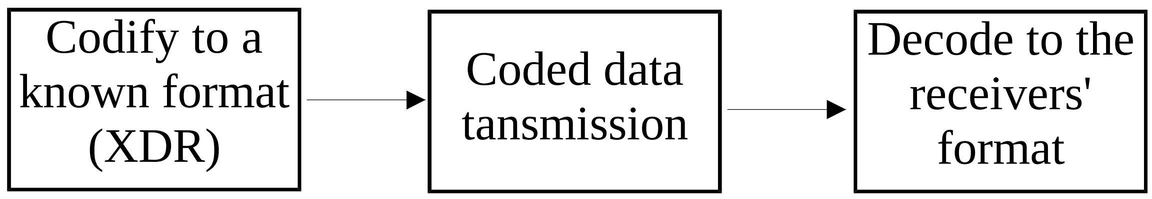
Codify to a known format (XDR)
Coded data tansmission
Decode to the receivers' format

Figure C.4: Strategy of De/Coding of PVM/MPI.

That is, before transferring data from one station to the other, data are coded in order to keep the information they represent. In this sense, both PVM and MPI are similar, if there exists data codification to be delivered, this codification is carried out before doing the transference.

In some way, this coding-transmission-decoding strategy can be considered conservative or anticipatory to the communication, because it always works independently of the communicated computers and is carried out before data are being delivered.

C.5.2 Direct Translation of Data Representation

One alternative to PVM’s (and MPI’s) anticipating feature of keeping the data consistency between the different computers consists in delaying the de/coding so that:

- Data are not always sent without some type of previous encoding (neither to XDR nor to any other format).
- Data are received as a bytes sequence in the receptor, together with a descriptor of the data type they represent plus the type of origin architecture (type of origin representation).
- If the processor in which the receptor process is located has a different data representation from the processor in which the sent process is located, data are “decoded”: the representation of origin data is changed so that it matches the target representation.

In consequence, there is no de/coding; rather, in the target processor, the communication routines will be in charge of “translating” the data representation of a computer (from which the message was sent) into the other (in which the message is received).

In the case of coding a known representation, there are two translations: that of the origin representation (from which the message is sent) into the known representation, and then from this into the target representation (where the message is received). In the case of direct translation, there is only at least one, which changes from the origin data representation to the target data representation when necessary.

Notice that the direct translation should be done in the target and not in the origin of each message, since in this way we avoid problems with the so-called collective communications, such as *multicast* and *broadcast*. In this type of communications, from the same process in a processor, we can send data to multiple processes (which potentially implies target processors), and thus there would not exist a unique and possible translating technique.

Direct translation for user processes can be as transparent as XDR encoding used in PVM and in (some implementations of) MPI. In the case of PVM, there are no drawbacks for its implementation because each computer of the parallel machine counts with identification functions as well as a means for identifying the location of each task (processor in which it is run).

As regards the codification of data for their transmission, translation has the advantage of minimizing the processing and memory used for each message on the side (processor) of the sent process. In fact, the sending process takes the data from its memory, adds the data descriptors (of how data are represented), and sends the message to the receptor process. On the side (processor) of the receiving process, both memory and processing depend on the complexity of the data translation. As it will be seen next, at least in the particular context of computer networks, the translation of representation is quite simple, and in consequence the need of memory and processing is also reduced.

Going a step further in the analysis of number representations in the particular context of computers, a surprising homogeneity is found as regards the acceptance of the standard IEEE 754 [6] for the representation of floating point numbers. In all the computers to which there is access and over which the experimentation was carried out, which includes

- PCs with Pentium processors (in some of its multiple versions), Celeron, and AMD K6–II,
- Sun workstations with processors MicroSPARC-II,
- An IBM RS/6000 workstation, with processor PowerPC,
- the adopted floating-point representation (at the level of operations of the processor’s floating point unit) is the same: IEEE 754 in both versions (single and double).

It is worth mentioning that the heterogeneity at the data representation level can be greater in other environments, for instance, among different traditional parallel machines with highly complex floating-point units. Independently of this, it should be recalled that the tendency (even in the most powerful/expensive parallel machines) is to use standard hardware. IBM SP2, for instance, are based on PwerPCs, and ASCI Red and Blue are based on Pentium Pro and MIPS R1x000 respectively.

Even when all computers adopt IEEE 754 as their floating point number representation, this does not mean that, when sending a byte sequence from a computer to another, these bytes will entail (represent) the same in both machines. Another hindrance is the way to save bytes of a particular type of data (in the case under analysis: floating point numbers) in memory. In this case, the “norms” followed are two, and, in fact, there does not exist many alternatives to explore: first the most significant byte (usually called *little endian* in literature), or first the least significant (big endian, in literature). It is clear that the translation of a format into another is immediate and without large memory or processing requirements.

From the last two paragraphs it is deduced that, as a minimum for the representation of floating point numbers, the direct translation of data representations among computers is advantageous in relation to codification, both in memory and processing requirements. It is not hard to make a similar analysis with the other types of basic data (characters, integers, etc) and arrive at the same conclusion. This is why the experimentation includes this way to “codify” data for the message transmission among pingpong processes.

C.5.3 Data Routing of a Message in PVM

Message routing in PVM refers to the way in which data of a message are transported among user processes and the PVM process in itself (*pvmd*) in each computer. The two most popular message routing ways among tasks are shown in Fig. C.5.

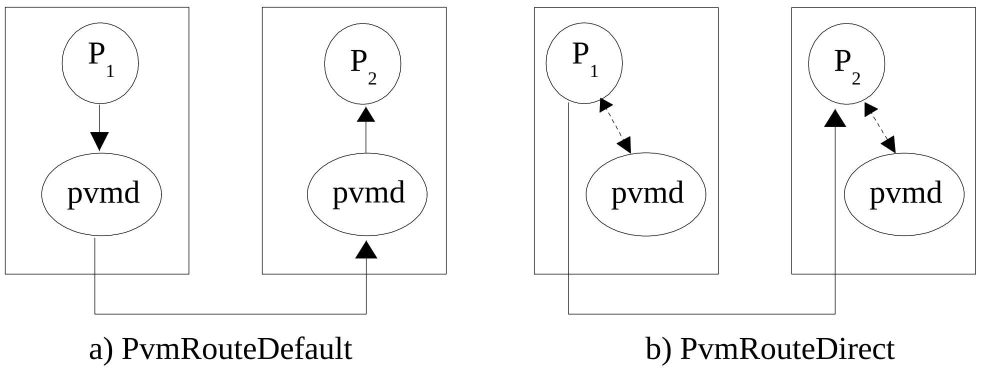

Figure C.5: Routing Basic Alternatives in PVM.

In the case of a) *PvmRouteDefault*, data are transferred among processes *pvmd* over the computer interconnection network using the UDP communication protocol (over IP).

In the case of b) *PvmRouteDirect*, data are directly transferred among the processes of the parallel application over the computer interconnection network using TCP communication protocol (also over IP). The general recommendation made in the PVM documentation is that, in performance terms, option b) is the best.

C.6 Different Ways of Message Transmission with MPI

Since MPI is proposed as a standard library of message passing among processes of a parallel application, option details such as encoding and routing (in the PVM sense) are not available at user levels. In this sense, the creation of MPI is quite far from and independent of computer networks and, in consequence, heterogeneity in data representation or in the alternative ways at communication protocols levels among processors do not have the relevance acquired from the beginning in PVM, which was created for (heterogeneous) computer networks. In terms of the possible implementations, it is highly demonstrated that MPI is possible within the whole range of message passing parallel computers, be it multiprocessors, multicomputers with *ad hoc* interconnection networks for parallel computing, or computer networks.

MPI implementations for computer networks generally assume that the data representation is heterogeneous (in fact, there are no many alternatives) and IP connectivity. The *advance encoding* method (with XDR, for instance) is usually used for solving the heterogeneity of data representations. As usual, in the context of MPI, we should recall that the implementation is in charge of making a decision of this kind, and, thus, different implementations of MPI can have different ways of implementing it. Similarly, what in PVM is referred to as routing (in reference to the way in which data of a message are sent from the sending process to the receiving process), depends on the MPI implementation, even though in most of the cases there is a tendency to use IP connectivity (at least those free that can be obtained via Internet).

In any case, independently of the MPI implementation, be it for computer networks or any other type of parallel computing architecture, there are no alternatives at user levels neither over the encoding or message data routing. In MPI, we gain in transparency with respect to the implementation and the parallel architecture, and in the particular case of computer networks, we perhaps lose performance if machines are homogeneous or if the previously explained data type representation translation is used.

C.7 Initial Experimentation with PVM

The initial experimentation with PVM was carried out in order to obtain a computing base of α and β using the different ways of encoding and routing that can be selected in PVM at user level. The machines with which the experimentation was carried out are detailed in Appendix X: CeTAD computers, interconnected with a 10 Mb/s Ethernet network. Since the computers **cetadfomec1** and **cetadfomec2** are exactly the same, the results are shown for one of them (**cetadfomec1**), which is referred to as **cf1.**

The pingpong method was used locating process *ping* always in the same computer and measuring the times with process *pong* in each of the rest, each time. Even though the most important to estimate (and thus, to measure) are parameters and , in the specific case of computer networks, the time necessary to solve the data representation heterogeneity (de/coding or translation) might be useful. The computer chosen to locate process ping is **purmamarca**, since it counts with the necessary memory quantity for all the message lengths (64MB) and is the fastest of the CeTAD computers.

Measurements were carried out with the network free of interferences (without much traffic in the network other than that generated by computers during the pingpong process), and they were actually carried out on different days under the same conditions, obtaining the same results.

According to what has already been explained, in heterogeneous computer networks, there exist four alternatives for the management of messages among processes:
1. *PvmRouteDefault* Routing with PvmDataDefault Encoding, i.e. transmit codified data in XDR format using process PVMD in each of the involved computers.
2. *PvmRouteDefault* Routing without codification, but with data representation translation, i.e. translating the representation whenever necessary in the target process and using process pvmd in each of the involved computers.
3. *PvmRouteDirect* Routing with PvmDataDefault Encoding, i.e. transmitting codified data in format XDR directly among pingpong tasks.
4. *PvmRouteDirect* Routing without codification but with translation of data representation, i.e. translating the representation whenever necessary in the target process and with the data transferred directly among *pingpong* tasks.

Since it is difficult to find a set of message lengths representing all the possibilities of applications and parallelization, the chosen lengths cover a wide range: from messages of eight bytes (the necessary for two single precision floating point numbers or one double precision) up to messages of 10^7 bytes. The intermediate lengths are 10^2, 10^3, 6×10^4, 10^6 and 10^7. The particular case of length 6×10^4 (which does not follow the “logarithmical convention” of the increase in message length) was chosen in order to then compare the results with the *pingpong* version of the operating system (in particular, Linux): the *ping* command, whose utilization is justified in the next section. Since length 6×10^4 is really too close to 10^5 to contribute with significant data, it was not included in the results.

For more information and clarification of the data obtained, the results will be presented in two formats:
1. Total communication time for the pingpong. This alternative is in turn presented in logarithmical format of the times, given the chosen message lengths.
2. Bandwidth or bytes/second, for a better idea of the performance relation between what is obtained at user level with PVM and the hardware (10 Mb/s Ethernet).

C.7.1 Performance with Routing Between pvmds and Codification

It is assumed that this alternative is the least convenient as regards the performance of the computer interconnection network, and the results obtained in terms of communication (pingpong) times are shown in Fig. C.6. It can be clearly seen that the communication latency time in PVM among processes varies between 1 and 10 ms depending on the computer, since these values repeat themselves in spite of the fact that message lengths vary between 8 and 1000 bytes (two orders of magnitude, in terms of growth).

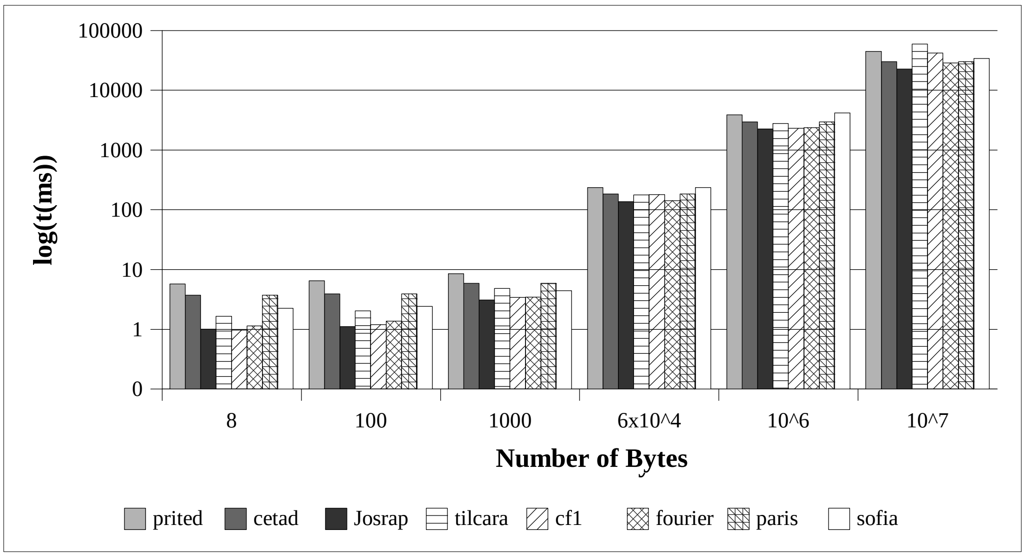

Figure C.6: Times with Routing and Encoding *PvmDefault*.

The logarithmical time scale of Fig. C.6 masks, in some way, the differences among computers, though we can see that, for all the lengths, there are different communication times for the different machines involved.

Fig. C.7 shows the same results, but in function of MB/s (megabytes per second), i.e. the quantity of information (2^{20} bytes) per time unit. Both in this figure and the following ones expressed in terms of MB/s, the relative differences among computers can be clearly seen, as well as the relation to the hardware capacity. In the particular case of this figure, what was expected can be proved for all the computers: the larger the message length, the higher the performance.

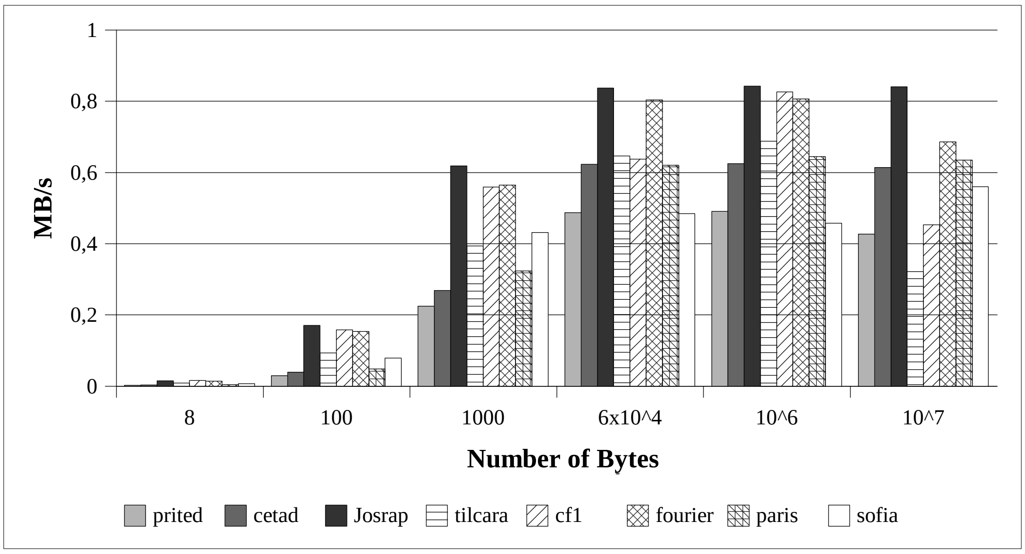

Figure C.6: MB/s with Routing and Encoding *PvmDefault*.

In the particular case of **tilcara** and **cf1** there seems to be an abnormal behavior for messages of 10^7 bytes, but this can be explained in function of their 32 MB main memory sizes. This size is not big enough for the message, plus PVM buffers, plus the rest of the processes (pvmd, operating system, etc.) and their memory requirements. In both cases we have to recur to the *swap* space (handled by the operating system), and in both cases this also produces a remarkable degradation at the level of the performance of user processes. In a certain way, **prited** has also the same problem (loss of performance due to its 32MB of main memory), but the relative loss is much lesser since in no case it exceeds 0.5 MB/s. The rest of the computers have a main memory of 64 MB or more and, thus, there is no need to recur to the *swap* space.

Also, from Fig. C.7:

- There are clear performance differences among the different computers, between slightly more than 0.4 MB and slightly more 0.8 MB/s.
- The best obtained is slightly more than 0.8 MB/s for three of the computers (taking into account **cetadfomec2**, represented by **cf1**)
- The best performance obtained for each of the computers is accomplished with message length of order of 10^4 bytes or more (represented with $6x10^4$, in the figure).

C.7.2 Performance with Routing Between pvmds and Representation Translation

Fig. C.8 shows the results obtained in terms of communication absolute times. Comparatively with those shown in Fig. C.6, the results of this alternative are similar in general terms: latency and absolute times, all of which gives us an idea of the relatively low importance of the way data are coded (or the representation translation, in this case).

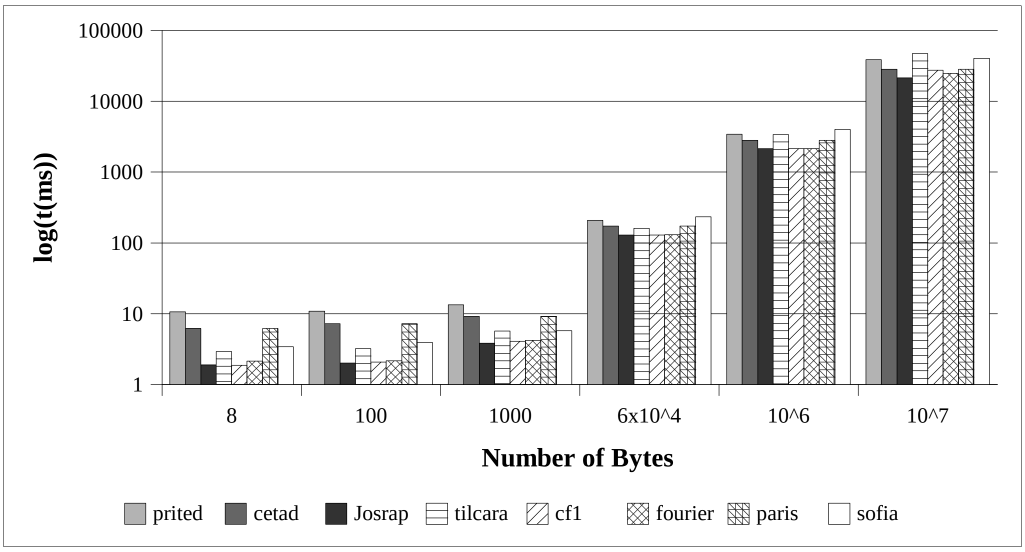

Figure C.8: Times with *PvmDefault* Routing and Translation of Data Representation.

Fig. C.9 shows the same results but in function of the MB/s according to the communication times and the quantity of bytes transmitted in relation to the message length. It can be noted that the decrease in the memory requirements make the 32 MB machines (**tilcara**, for instance) performance not fall so drastically when messages are of 10^7 bytes, because they make lesser use of the *swap* space and, thus, the impact on the memory handling speed is smaller.

In addition, it can be observed that, as direct impact of the representation translation, performance of machines communicating the same data representation can be even better. Since process *ping* is always assigned to a PC with Linux (**purmamarca**), the performance is better with the rest of the PCs (**tilcara**, **fourier** and **cf1**), in comparison with the previous alternative.

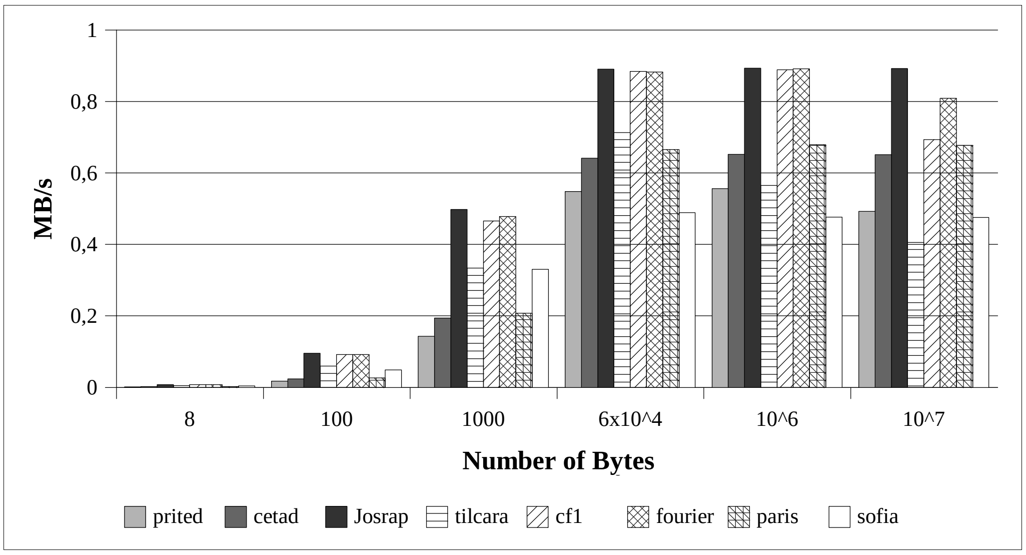

Figure C.9: MB/s with *PvmDefault* Routing and Translation of Data Representation.

Perhaps, the most significant point in both cases (XDR codification and representation translation, which is the only thing that has varied until now) is the really low performance of computer **sofia**, which does not reach 0.6 MB/s in any case. This low performance is stressed when it is compared to computers whose relative processing speeds are several times slower, like computers **prited**, **cetad** and **paris.**

C.7.3 Performance with Routing Between User Tasks and Codification

Fig. C.10 shows the communication times obtained when user tasks communicate directly using the TCP protocol, and with XDR (PvmDataDefault) data encoding. This alternative of message transmission is what the PVM documentation recommends when computers (and their ways of representing data) are heterogeneous.

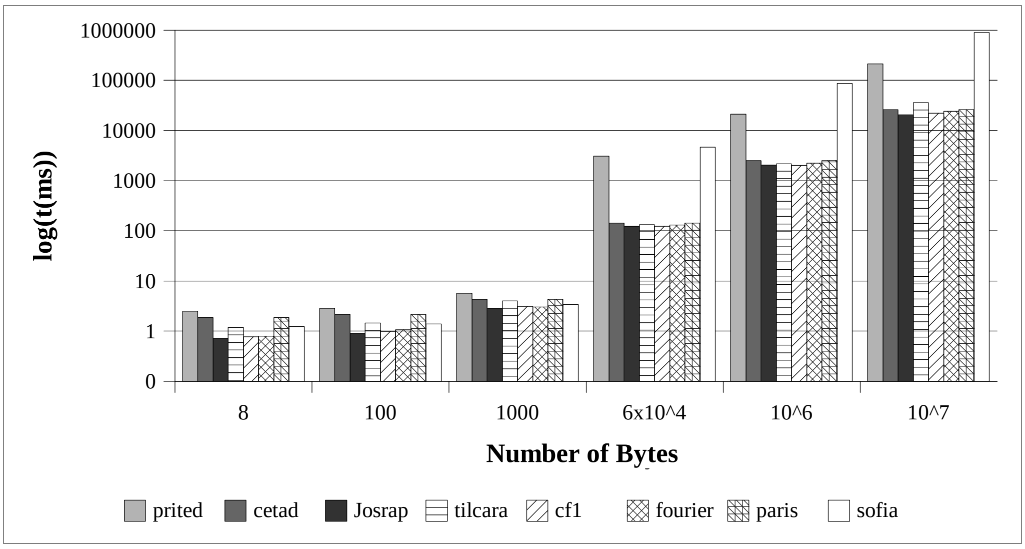

Figure C.10: Times with Routing among Tasks, and *PvmDefault* Encoding.

Comparing the results of Fig. C.10 to the previous, there are some slight differences, such as:

- the latency time is lesser (closer to 1ms than to 10ms),
- in logarithmical scale, for small message times (up to 1000 bytes), computers do not present significant differences. Once again, it should be noticed that the logarithmical scale masks many differences that can be then clearly visualized in terms of MB/s.

Surprisingly, Fig. C.10 shows the excessive communication time for messages greater than 1000 bytes in **prited** and in **sofia**. Even in logarithmical scale, the difference is remarkable, and it is really interesting that, in principle, it appears with two unrelated computers (in terms of design) in terms of software or hardware; **prited** and **sofia:** one a Sun SPARCStation 2 from the beginnings of the '90s and the other, an IBM RS/6000 from the end of such decade.

The last of the listed conclusions has too much relevance, since it thoroughly contradicts the documentation and reports related to PVM. In this sense, the PVM installation, the *pingpong* programs, and the potential reasons for which this might have happened were verified, though no significant reason was found in relation to the PVM library.

Similar tests of TCP connections between computers **purmamarca** and **sofia** showed a similar performance at user process level. This implies that PVM communication routines show nothing but what occurs at connection level between computers. The similarity (in terms of communication performance) between computers **sofia** and **prited** makes us assume the same behavior at TCP connection level.

Fig. C11 shows the same results but in function of MB/s. Now the differences among computers can be more clearly seen, specially with message lengths greater than 1000 bytes.

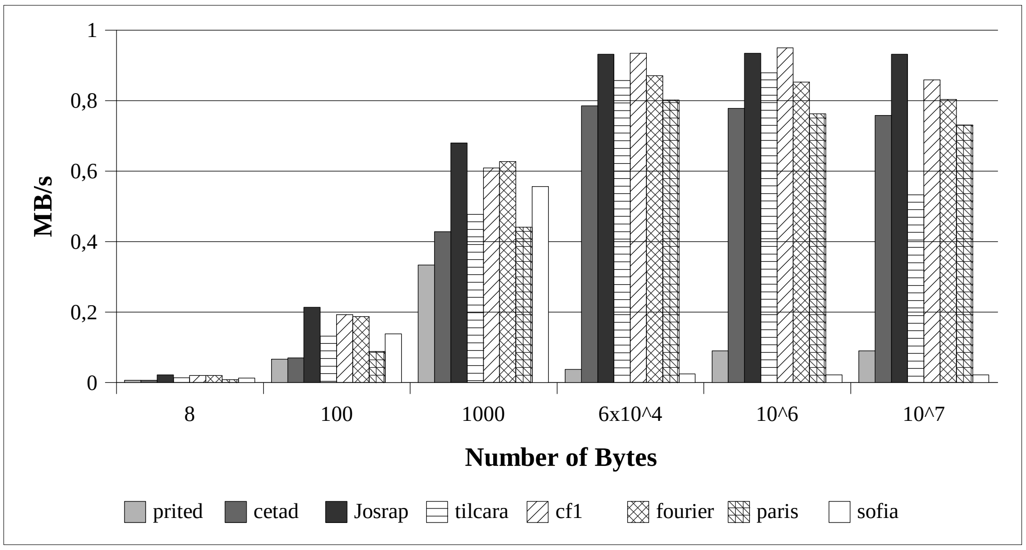

Figure C.11: MB/s with Routing among Tasks, and *PvmDefault* Encoding

C.7.4 Performance with Routing Between User Tasks and Representation Translation

Fig. C.12 shows the communication times obtained for the Routing of tasks (TCP) and with data representation translation.

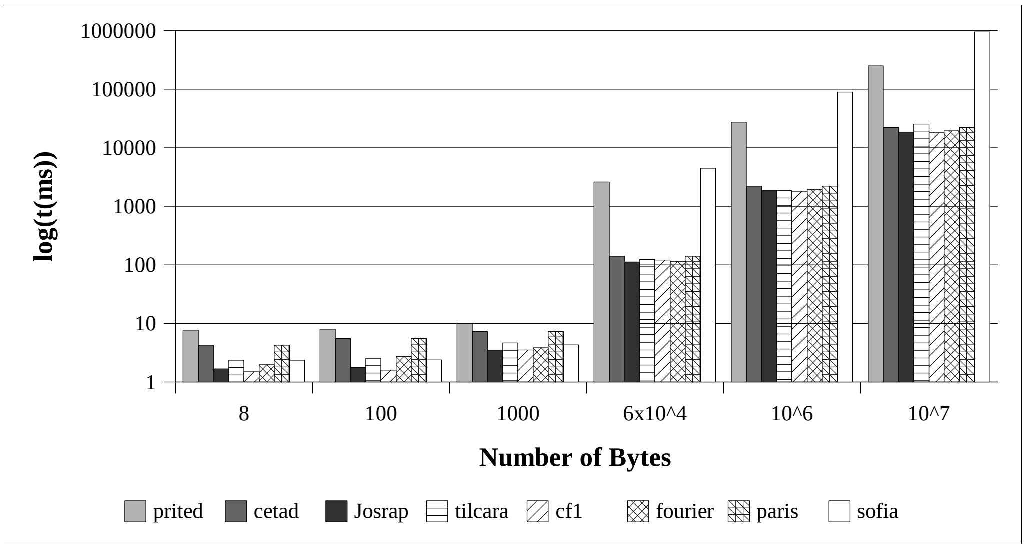

Figure C.12: Times with Routing among Tasks and Translation of Representation.

Fig. C.13 shows the same results in function of MB/s obtained among user tasks communicated to the PVM functions, except as regards the translation of data representation, which is independent of PVM.

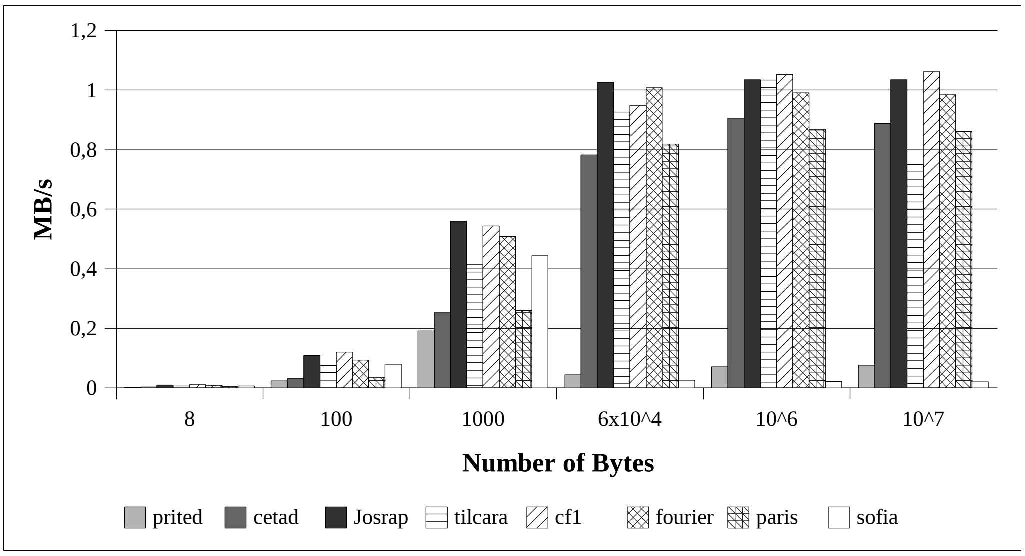

Figure C.13: MB/s with Routing among Tasks and Translation of Representation.

With this alternative, and for the first time, 1 MB/s of bandwidth is outperformed for communications among computers with the same data representation. Both **purmamarca,** where process *ping* is always allocated, and **Josrap**, **tilcara**, **fourier** and **cetadfomec1** (and **cetadfomec2**) are PCs (though with different processors) with Linux operating system. On the other hand, messages can be sent with PvmDataInPlace, which implies no copy in buffers, nor codification that may increase the final size of data to be transmitted over the computer interconnection network.

The information given by these figures is very similar to what the two previous show (Fig. C.10 and Fig. C.11), both in absolute values and the relation existing among computers. It also shows (and, in this sense, proves in some way the previous results) the great difference in performance existing between the computers **prited** and **sofia** with respect to the rest.

The very low performance of **prited** and **sofia** with this communication alternative (TCP routing of tasks and translation of representation), among PVM tasks, makes it possible to definitively discard that the problem might be codification. On the other hand, the problem seems to be the TCP communications and/or PVM communications with TCP routing.

C.7.5 PVM Experimentation Conclusions

If it is expected communications to the maximum possible performance, computer **sofia** should be specifically analyzed, since it should have a much higher communication performance. In the particular case of messages with direct routing, the performance obtained is actually much lower than the expectable and, thus, it is possible to find an error in the way TCP communications are handled in the operating system or in terms of some computers' TCP connection configuration (at operating system level). In this sense, the experimentation did nothing but show that there might be a (serious) problem in the performance due to a software problem.

In addition, in the context of the performance, it is interesting to recall that *all* the computers have the same performance as regards communication hardware. This means that all the computers, independently of the manufacturer, have Ethernet 802.3 interface (NIC), and thus they all would be capable of communicating at 10 Mb/s, which implies in theory 1.25 MB/s (1.25 x 2^{20} bytes per second). We can suspect certain sustained performance loss given by the overhead imposed by the operating system and its buffers, processes, etc., plus all related to PVM in itself; however, we do not count with an *a priori* quantified idea of the performance loss implied by all of this overhead. In consequence, knowing the maximum possible performance of communications for user tasks may always remain pending if the performance is only monitored with the pingpong method with tasks using PVM.

Going back to the initial objective of α and β value estimation, there are some very important conclusions: the range of the values and the heterogeneity of the values.

- Depending on the message communication alternative chosen, latency varies between 1 and 10 ms.
- Also depending on the chosen alternative (and excluding the particular cases of **prited** and **sofia**), the data transference rate (or bandwidth) ranges from little less than 0.5 MB/s and little more than 1 MB/s.
- Whatever the chosen alternative is, both the latency and the transfer rate depend on the computer.

The latter conclusion is quite discouraging because a subsystem, which is homogeneous in theory like that of the computers' interconnection, "becomes" heterogeneous when it is regarded from the point of view of user tasks which make up a parallel application.

The immediate consequence of communication time heterogeneity in Ethernet networks with PVM is: a message that should reach any process in other processor in the same period of time, it will now reach it in a period of time which depends on the origin and target processors, beyond the solution adopted to solve differences in data representations.

In order to prove the results obtained with PVM, a new set of experiments is designed to improve precision as regards:

- α and β for user processes;
- heterogeneity, or not, of communications;
- the particular cases of low performance of **prited** and **sofia**.

C.8 Experimentation with the Linux ping Command

The ping command (at least in its *version* of Linux operating system) is versatile enough to:

- Measure communication times at user application levels.
- Generate "messages" of different lengths.

In fact, any user without special permission can run the *ping* command, which generates a user process that produces a *pingpong package* per second at ICMP (Internet Control

Management Protocol) protocol level. The package round-trip time is reported by the *ping* command itself, reason why no instrumentation is added. The *pingpong package* length can be varied using a command option, and this in turn allows us to watch the interconnection network performance in function of the quantity of transferred data, considering the *pingpong package* as a message.

Fig. C.14 shows the communication times involved for different number of bytes containing the *pingpong packages*.

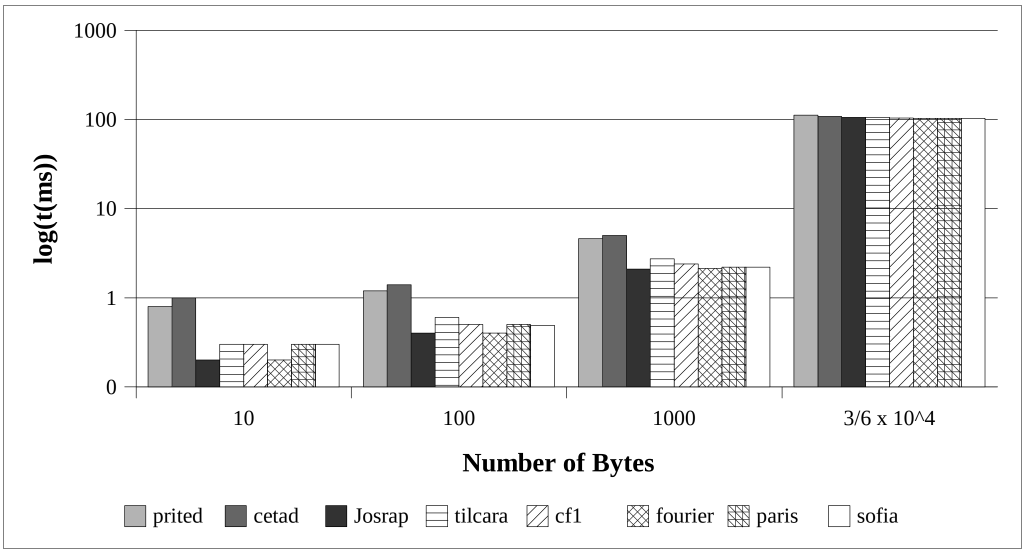

Figure C.14: Linux ping Times.

It is necessary to make some explanations as regards the *pingpong packages* lengths as well as the last identifier appearing as "3/6 x 10^4". The whole ICMP package must be smaller than 64 KB, like IP packages, and thus what can be measured with the ping command reaches this transferred byte quantity.

In the particular case of computers with Sun 4.1.x (BSD-based Sun OS) operating system, **prited** and **cetad** - for some reason undocumented -, do not work on ICMP of more than 32 KB and thus, with these computers, pinpongs were carried out with a maximum size of 30000 bytes (then multiplying the time by two in order to equalize them with the times of the rest of the computers).

From the information gathered in Fig. C.14:

- The messages latency time (α) for user tasks (which communicate themselves with the ICMP protocol) is of order 1ms. Like in PVM, the latency time depends on the computers, though the variation range is quite smaller.
- Just after messages of 1000 bytes or more, time starts to be proportional to the data quantity transferred. This means, within this context, that up to that "message" length (in this case, pingpong packages), the latency times is large enough to be greater than the data transmission time.

Fig. C.15 shows the same results but in terms of MB/s of *pingpong* package data transference. We can clearly notice the differences between the computers partially masked by the logarithmical scale of the times of the previous image.

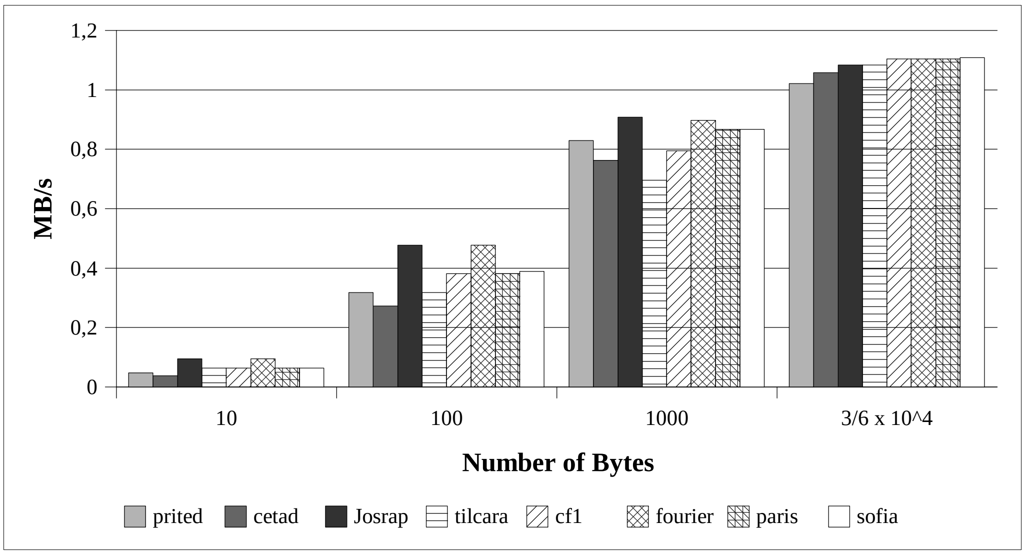

Figure C.15: Linux ping MB/s.

As with latency, the data transfer rate is not exactly the same for all of the machines, but the variation range is much smaller than that obtained when processes communicate themselves in PVM. In addition, in terms of data transference rate:

- The maximum obtainable with ICMP is almost the physical, with which the difference between this maximum (with the ping of Linux) and that obtained in PVM is due to the PVM overhead.
- The particular case of low performance of computers **prited** and **sofia** is not due to hardware or the protocols closer to the hardware (like IP and ICMP).

Like with the experimentation with PVM, just after message sizes of order 10^4 bytes (30000 bytes for **prited** and **cetad**, and 60000 for the rest), a better performance of communications is obtained. However, with these experiments, important information is added complementing the contributions of the initial experimentation carried out with PVM.

C.9 Conclusions of PVM and Linux ping Experimentation

From the results obtained in PVM and with the ping command of Linux, it is worth attempting an intermediate. It is very likely that we will not get a performance exactly the same as the observed with the ping command because:

- With the ping command, differences in data representation are neither taken into account nor solved.
- Communication routines outperforming a 64 KB message length must be implemented.

In other words, and unfortunately, not always the messages to be transferred between processes can be encapsulated in a single communication protocol package closer to the hardware like ICMP.

However, it does not seem that these two constraints, though strong, will make PVM have:

- Such low communication performance, like in the case of **prited** and **sofia** with direct routing.
- Such heterogeneous performance depending on the computers among which data are transferred.

In some way, on the one hand, the ping command seems to give a really optimist version of the communication network performance, and on the other, PVM seems to have a really high overload. This large PVM overhead produces a high decrease of the interconnection network performance and this reduction of the performance is proportional to the computer's relative speed. In consequence, the communication performance starts depending on the communicating computers and as heterogeneous as the computers' heterogeneity.

Due to this, it is natural to seek an intermediate solution as regards the performance of the computer interconnection network. This implies having better and more homogeneous performance at the level of user parallel application processes (according to the communication hardware) than that obtained with PVM, though sometimes it is not the obtained with the ping command.

In addition, it is necessary to recall that, even though the whole experimentation (and thus the conclusions reached at) is based on the *pingpong* method, i.e. on point-to-point communications between two and only two processes:

- the original problem to solve is that of matrix multiplication,
- the algorithm developed to multiply matrices in computers networks is based on broadcast,
- it is important to count with an efficient implementation of broadcasts due to their wide use in parallel programs [11].

As it can be concluded from the experimentation shown in this chapter, the point-to-point communication performance with PVM is not fully satisfactory. Consequently, there is no reason to assume that the performance of collective communications, which are the most important for the application under analysis and for many others, is indeed satisfactory.

For the above reasons, the next chapter will explain the development and the performance obtained in principle for a collective broadcast communication routine that can be extended to a collective communication library.

C.10 UDP-based Broadcasts

The main performance objectives for the development of a broadcast routine different from that provided by message-passing libraries like PVM and MPI implementations are:

- Improving the performance obtained from the user processes in relation to that obtained with "general purpose" libraries, such as PVM.
- Obtaining performance homogeneity according to the hardware homogeneity, which, in the case of PVM, is not verified in the experimentation.

In addition, using PVM, the ways of sending the same message to more than one target process are two:
- Multicast routing, **pvm_mcast().**
- Broadcast routing in a group, **pvm_bcast().**

And the implementation of both routines is based on multiple point-to-point messages. That is, both pvm_mcast() and pvm_bcast() imply that, as minimum, the same message is sent *m* times from the origin computer (where the process sending the message is being run) to the *m* machines where there is at least one target process of the message. If, for instance, a broadcast or multicast message has five receptors and each of these receiving processes is being run in a different machine (and different from the machine where the process sending the message is being run), the total time of the message will be approximately equal to five times the time of the same message, as if it were sent to another process run over another machine. For these point-to-point communications among machines, the same routines with which the experimentation was carried out are used.

In principle, a routine for broadcast messages with *acceptable* performance in Ethernet networks is required, and where acceptable can be defined according to:
- Communication time absolute values closer to that provided by the hardware than those observed in the experimentation with PVM.
- Scalability in terms of machine quantity, since when we take advantage of the Ethernet network broadcast capabilities, the same message can be sent and received to/in as many computers as are connected. It is evident that there will be a penalization depending on the quantity of computers receiving the same message due to the synchronization and maintenance of the transferred data reliability; however, this penalization should be far from the repetition of the same message as many times as different computers should receive it.

Both the performances of pvm_mcast() and pvm_bcast() are not acceptable and, thus, the PVM library would be of no use for broadcast messages requiring the matrix multiplication algorithm in parallel. At this point there are several alternatives, and the two most important are:
1. Using another message-passing library, such as some MPI implementation, which is usually focused for this type of parallel architectures.
2. Implementing a broadcast message routine (and, eventually, a whole library of collective communications) in order to make explicit use of the Ethernet networks' broadcast facility.

The use of a message-passing library has, in principle, a fundamental drawback from the point of view of the performance or of "prediction" of the good performance of broadcast messages. In the specific case of MPI, it is clear that the performance depends on the implementation. More specifically, the implementation will be what determines the degree of utilization of the Ethernet network characteristics for broadcast messages. In this sense, MPI and, in particular, all its implementations share some degree of uncertainty, in terms of broadcast messages performance, with the rest of the message-passing libraries, including PVM. The difference in this case are the specific experiments which were carried out and determined the characteristics of the broadcast (multicast) messages performance for PVM and not for the rest of the libraries. In fact, it is rather hard to optimize message-passing libraries to meet the characteristics of the Ethernet network since:

- In general, libraries are, one way or the other, proposed as standards for message passing parallel machines and, thus, it is pointless to orient them to a specific type of interconnection network. In fact, both PVM and MPI have been implemented for different types of parallel machines and, hence, it is pointless to orient them a priori to Ethernet interconnection networks.
- In general, libraries provide a large quantity of communication routines. Event though in theory it can be asserted that with the primitives send-receive for point-to-point processes communication are enough, it has also been concluded that there exist a wide range of communication routines considered useful and even necessary in some cases. Perhaps, the clearest example with this respect is the very definition of the MPI standard. In this context, it is very hard to orient or optimize one or one type of communication routine for one or one type of interconnection network without producing an excessively expensive and/or too specific library (in terms of development, maintenance, etc.).

For these reasons, broadcast message routine between user processes with a set of design and implementation premises has been chosen to be implemented, so that:

- It takes advantage of the very broadcast of Ethernet networks, and in this way, it is optimized in terms of performance. Since the algorithm exclusively depends on broadcast messages, when the Ethernet network broadcast is used up, there is a really good expectation in terms of scalability because the communication time is expected to remain constant and do not increase proportionally to the quantity of computers used.
- It is simple enough not to impose a too heavy load in terms of implementation and maintenance. In addition, it is clear that simplicity per se largely contributes to the optimum performance. On the other hand, the proposal is specific enough to make the implementation simple.
- It has the maximum possible portability, in order to be used, whenever possible, even in the context of other interconnection networks (leaving aside Ethernet).
- It is implemented and installed from the user mode, without changing the operating system (the kernel) and without needing special permissions (superuser). It is normally accepted that the best results in terms of performance are obtained adapting the kernel and/or with the possibility of managing the process priorities, such as in [5] [4] [12]. These possibilities are discarded since:
    - In general, free-use libraries do not use these characteristics, and thus it would be like changing the parallel software development context. Basically, a user who has always used PVM never had/have to neither obtain special priorities nor change the operating system in itself.
    - The original proposal is directed to installed computer networks and each computer does not thus necessarily have as single or main objective parallel computing. In fact, we may find the case of different administrators for each of the computers to be used in parallel and this produce, at least, a multiple administration task which, in general, is not easy to solve.
- It could eventually be extended to a whole collective communication library, such as

those proposed by [2] [1] [3], though specifically oriented to Ethernet interconnection networks.

Most (if *not all*) the previous premises are fulfilled when all the design and implementation of the broadcast routine is based on the UDP protocol standard (User Datagram Protocol) [8] over IP (Internet Protocol) because:

- UDP allows sending a same data or set (package) of data to multiple targets at user application level.
- Such as verified in all the machines used, the UDP protocol implementation takes direct advantage of the Ethernet network broadcast capacity.
- In principle, it seems reasonable that the broadcast directly implemented as part of the UDP protocol has a better performance than the implemented by a user. If in an ATM network, for instance, it is possible to use UDP, it is very likely that the UDP broadcast will be better (in terms of performance) than that potentially implemented from user processes. Even though the performance is not taken into account, whenever there exists a UDP protocol implementation we will be able to use the proposed broadcast, independently of whether the interconnection network is Ethernet or not.
- The user interface provided by the sockets is simple enough and widely extended to all the UNIX versions, so as to simplify the implementation of the broadcast routine, even when problems related to process synchronization (in the same or in different computers) and communication reliability are to be solved.
- UDP, TCP, and IP protocols are easily usable from the user processes, at least in the standard computers of the installed local networks.

In brief, a new routine of broadcast messages based on UDP and portable to, at least, all the UNIX versions used in all the local networks in which the experimentation is carried out. With this broadcast messages routine the same experiments were carried out for the PVM communication routines. Initially, the results of point-to-point communications ("broadcast" or "multicast" message using a single receptor process) are presented; and finally, the results of broadcast messages with pvm_mcast() and pvm_bcast() of PVM and with the proposed UDP-based routine are also shown.

C.10.1 A Single Receiver (Point-to-point Messages)

In order to compare the results with those obtained by the experimentation with PVM point-to-point communication routines and with the Linux ping command, the broadcast message routine was used as single receptor process. The results referring to the communication times appear in Fig. C.16. It should be noticed that these communication times between two machines may not be optimal since the communication routine is designed for broadcast messages involving more than one computer receiving messages. All the same, it can be used for comparing the results shown in

- Fig. C.14 with the communication performance according to the ping command (ICMP protocol).
- Fig. C.12 with the PVM communication performance with Routing of tasks (TCP protocol) and data representation translation.
- Fig. C.8 with PVM communication performance with routing among PVM processes (pvmd, UDP protocol) and translation of data representation.

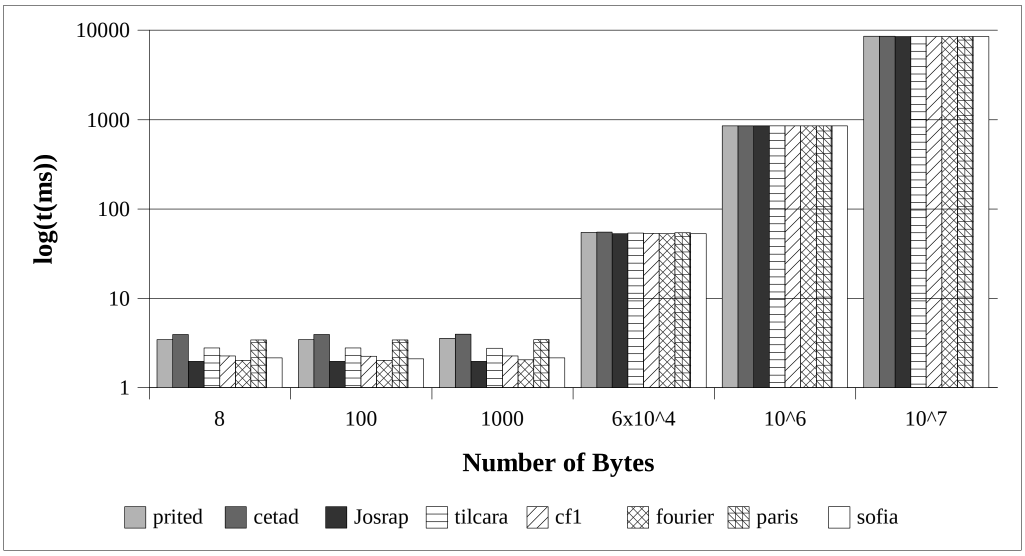

Figure C.16: “Point-to-Point” Times with UDP-based Broadcast.

Comparing these results with those of the Linux ping command (Fig. C.14):

- Latency is quite higher, and it is verified that, at least up to the transference of a hundred bytes, the latency time will be that dominating the total time. On the other hand, it is confirmed that the communication latency depends on the machines involved in the data transference.
- Communication time is similar to that obtained with the Linux ping command for 30000 (**cetad** and **prited** workstations), and 60000 bytes (the rest of the computers).

Comparing these results with those of the PVM point-to-point communication routines (Fig. C.8 and Fig. C.12):

- Latency is similar to that obtained with PVM, and in many cases is almost identical.
- Computers **cetad** and **prited** do not show any performance anomaly for any size of message. This confirms that the problem is due to the configuration and/or the implementation of the TCP protocol in these computers.
- At least from the message length of 6000 bytes, the performance of communications is homogeneous, such as expected from the point of view of the communication hardware.
- The performance with the UDP-based broadcast is rather higher than that obtained with the PVM point-to-point communication routines. Even with the logarithmical scale, communication times are really close to the optimal, as if there were no overhead of the different layers of the software involved (operating system, broadcast routine, etc.). For instance, the communication time of 10^6 bytes messages is really close to a second (or 1000 milliseconds, such as shown in Figure C.16).

Fig. C.17 shows the same results in terms of asymptotic bandwidth, where we can clearly verify that the results are highly satisfactory in terms of performance.

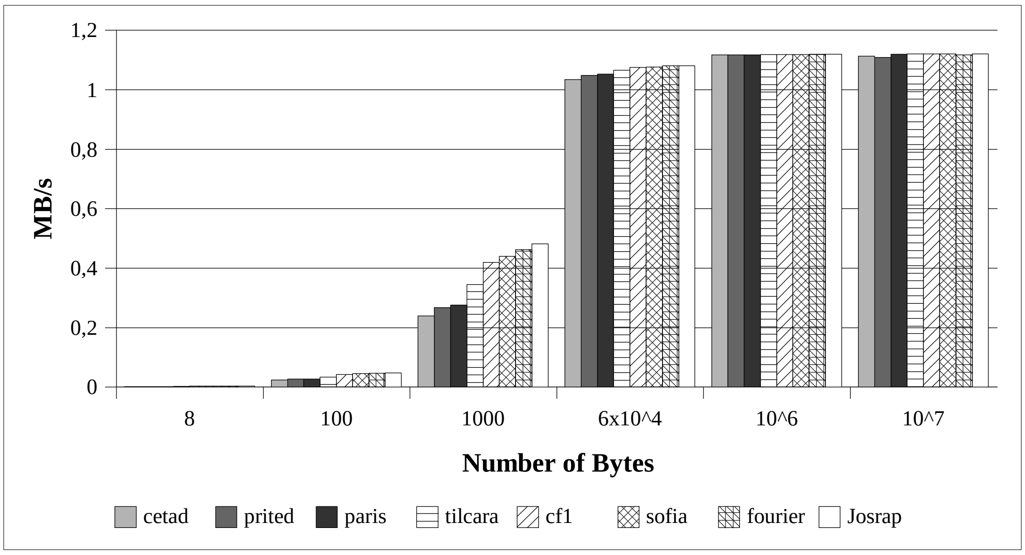

Figure C.17: MB/s "Point-to-Point" with UDP-based Broadcast.

With these results of point-to-point communications:

- Almost all of the communication hardware performance at the level of parallel application processes. The overhead of all the intermediate software layers does not almost affect the final performance among processes.
- In the absence of collisions, the performance is independent of the communication network and independent of the involved computers. The computer heterogeneity with their relative differences in terms of computing capacity is not translated, like in PVM, into "heterogeneous performance".

C.10.2 Broadcast Messages

The final objective of broadcast communications is not data transference from one process to another, but the transference from a process to a certain quantity of processes running in different computers. It is for this reason that we have to verify, at least with tests, that broadcast messages will be sent among processes with near-optimal performance and relatively independently of the quantity of receptor processes, or computers involved in broadcast messages.

Fig. C.18 shows the communication times involved for different lengths of broadcast messages and different quantity of receptors assigned in different machines. With the aim of comparing in a better way the different message lengths, we have chosen to show them in the same graphic instead of displaying a graphic for each message length. Separated with vertical point lines, over the axis **x**, the broadcast message times are shown for different quantity of receptors (machines). The PVM results shown are independent of the use of the routine pvm_mcast() or pvm_bcast(), because they have a similar performance.

It can be noticed that the time with PVM routines is kept quite independent of the message length of 8, 100 and 1000 bytes, respectively. In fact, it depends much more on the machines involved. For these message lengths, the proposed UDP-based routine uses relatively constant times and independent of the quantity of computers involved. For broadcast messages of 60000 and 1000000 bytes, the time is directly proportional to the quantity of receptors when the PVM routines are used. Once more, when the routine directly based on UDP, the time (apart from being quite better than the best of PVM) is relatively constant and independent of the computers involved.

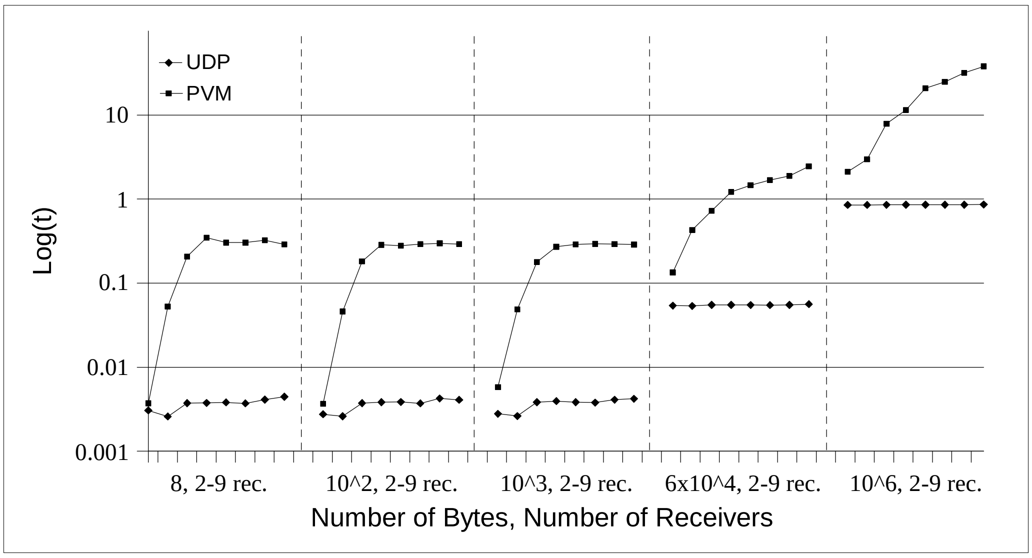

Figure C.18: Broadcast Times with PVM and based on UDP.

Fig. C.19 shows the same results but in terms of MB/s and, thus, some details are masked by the logarithmical time scale of the previous graphic. For the computing of the asymptotic bandwidth or transference rate or MB/s of a broadcast message, we should recall that the message is unique, the same data should reach multiple targets independently of the fact that the implementation is carried out with multiple point-to-point messages or in some other way.

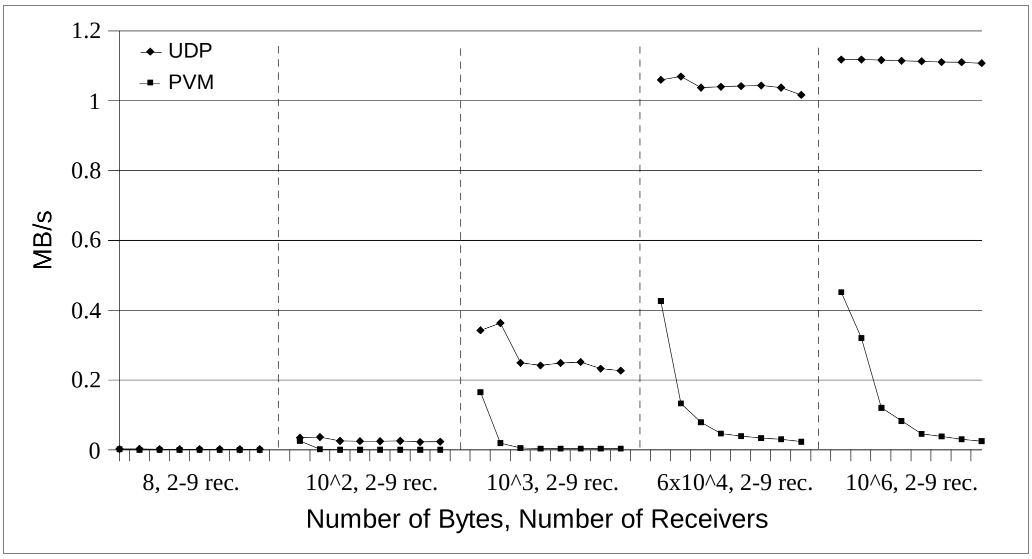

Figure C.19: MB/s of Broadcast with PVM and based on UDP.

As previously explained, the communication time is dominated by the latency, at least for messages up to 1000 bytes. In consequence, the performance in terms of bandwidth or transference rate is quite poor using PVM or the UDP-based broadcast routine. For messages of 60000 bytes, we can easily notice that the performance, when using routines provided by PVM, depends on the quantity of receptors and relatively constant, when using the UDP-based routine. The variations in this last case are basically based on the fact that, for this message length, the latency of each machine affects the total time of the communications. For messages of 10^6 bytes, the latency of each machine is already lesser than the data transference time and, thus, the performance has less variation in asymptotic bandwidth. In the case of PVM, once more, the performance is proven to decrease as the quantity of computers used increases and, thus, we can assert that the broadcast implementation uses multiple point-to-point messages.

C.11 Broadcasts in the LQT Local Area Network

Experimentation in LQT is similar as regards broadcast messages. The same characteristics appear: performance depends on the quantity of computers involved for PVM routines, and almost constant performance for the proposed routine directly based on UDP. Alike what occurs in CeTAD, heterogeneity is lower since, for instance, there are no differences in terms of data representation, because all available computers are PCs. Also, alike CeTAD, the quantity of computers is lower and thus the maximum performance degradation (when all computers are used) is lower as well.

One of the most remarkable differences yielded by the LQT experimentation in relation to that of the CeTAD refers to the point-to-point messages latency. Since machines are quite similar and performance differences as regards computing capacity are more restricted, message latency is much more limited in terms of absolute value range. In fact, this makes communication performance among user processes even more independent, since now not only is the asymptotic bandwidth closer to the interconnection network hardware's (using the UDP-based broadcast messages) but also the total communication time is much closer to what can be estimated with the hardware values.

C.12 Broadcasts in the LIDI Local Area Network

As expected in LIDI local area network, communication performance is better since the communication network is a 100 Mb/s Ethernet instead of 10 Mb/s, like those of the CeTAD and LQT. Even so, experimentation proves that for PVM the broadcast message time depends on the quantity of computers involved. It also proves that broadcast message time relatively depends on the quantity of machines involved when the directly UDP-based routine is used.

It is very interesting how latency influences the total communication time when the hardware bandwidth is ten times greater. Since most of the latency is due to the overload of factors external from the very Ethernet interconnection network (operating system, protocols, communication routines among processes, etc.), the absolute latency time is quite independent of the bandwidth capacity. Multiplying by ten the bandwidth of the Ethernet network, the latency time "weight" is implicitly multiplied (though no necessarily by ten) in the total time of the communications. The quantification of these factors should be computed at least with the design of specific experiments, which are out of the scope of this Appendix.

The impact of the latency on the final time of communications is definitely important taking into account the tendency to use networks with greater bandwidth capacity. All the same, it is expected that the latency at interconnection hardware level will be reduced proportionally to the bandwidth increase and that the overhead imposed for the user processes will be reduced.

References

[1] Bala V., J. Bruck, R. Cypher, P. Elustondo, A. Ho, C. Ho, S. Kipnis, M. Snir, "CCL: A Portable and Tunable Collective Communication Library for Scalable Parallel Computing", Proc. of the 8th International Conference on Parallel Processing, IEEE, April 1994.

[2] Banikazemi M., V. Moorthy, D. Panda, "Efficient Collective Communication on Heterogeneous Networks of Workstations", Proc. International Conference on Parallel Processing, pp. 460-467, 1998.

[3] Barnett M., S. Gupta, D. Payne, L. Shuler, R. van de Geijn, J. Watts, "Interprocessor Collective Communication Library (InterCom)", Proc. of the Scalable High-Performance Computing Conference '94, Knoxville, TN, USA, IEEE Computer Society Press, pp. 357-364, May 1994.

[4] Chiola G., G. Ciaccio, "Lightweigth Messaging Systems", in R. Buyya Ed., High Performance Cluster Computing: Architectures and Systems, Vol. 1, Prentice-Hall, Upper Saddle River, NJ, USA, pp. 246-269, 1999.

[5] Ciaccio G., "Optimal Communication Performance on Fast Ethernet with GAMMA", Proceedings Workshop PC-NOW, IPPS/SPDP'98, Orlando, FL, LNCS No. 1388, Springer, pp. 534-548, April 1998.

[6] Institute of Electrical and Electronics Engineers, IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Std 754-1984, 1984.

[7] Pacheco P., Parallel Programming with MPI, Morgan Kaufmann, San Francisco, California, 1997.

[8] Postel J., "User Datagram Protocol", RFC 768, USC/Information Sciences Institute,

Aug. 1980.

[9] Postel J., “Internet Protocol”, RFC 791, USC/Information Sciences Institute, Sep. 1981.

[10] Sun Microsystems, Inc. XDR: External Data Representation Standard. RFC 1014, Sun Microsystems, Inc., June 1987.

[11] Wilkinson B., Allen M., Parallel Programming: Techniques and Applications Using Networking Workstations, Prentice-Hall, Inc., 1999.

[12] GAMMA Home Page http://www.disi.unige.it/project/gamma/