LRP 537/96 February 1996

A CLIENT-SERVER-MONITOR DATA MANAGEMENT ENVIRONMENT FOR PARALLEL COMPUTATION

S. Wüthrich, S. Merazzi & T.M. Tran

A Client-Server-Monitor data management environment for parallel computation

S. Wüthrich[a], S. Merazzi[b] and T.M. Tran[c]

[a]Cray Research (Switzerland) S.A.,
Cray-EPFL PATP Center, PSE, CH-1015 Lausanne
[b]SMR Corporation, PO Box 41, CH-2500 Bienne 4
[c]CRPP-EPFL, 21 av. des Bains, CH-1007 Lausanne

Abstract

A *Client-Server-Monitor* computational environment is presented. This environment is based on a data management system which provides tools for the logical and physical organization of data. Calculations are performed by *Client* programs on a distributed memory system under the control of the *Server*. Data are handled by means of a *Data Base Server* module connected to the *Server*. The *Monitor* module, which is based on the XGrafix library [1], offers capabilities for on-line interactive monitoring of running applications. The MEM-COM Data Base [2] is used as data management system and communication within the *Client-Server-Monitor* environment is done using PVM. Performance measurements are presented as well as experiences with a PIC simulation running on a Cray T3D.

Introduction

Over the last decade supercomputers have become an increasingly employed tool in a large variety of fields in both academic and industrial institutions. Real life computer applications require however very large memory and fast processing units. A new generation of massively parallel processing (MPP) computers have been proposed by a number of supercomputer vendors in order to provide significant increase in both memory and computational power over the traditional vector or scalar supercomputers. These systems differ from each other depending of the memory configuration (shared or distributed) and the chosen processor type used as processing elements (PE). All the developments reported in this paper have been made for distributed-memory computer systems. A distributed-memory system is either a MIMD-type (Multiple Instruction, Multiple Data) of massively parallel computer like Cray's T3D or composed of separate processing units interconnected by a communication network such as a cluster

of workstations.

One of the consequence of the growth in power and speed of this new generation of supercomputers is the very large amount of data one has to deal with. On serial supercomputers, input and output data manipulation, on-line storage of information, post-processing treatment of results as well as data archival of large calculations are often handled by advanced tools such as a data management system (DMS). With the advent of massively parallel computers, DMS have to be extended in order to take advantage of the scalability and the larger capabilities offered by MPP systems. Due to the increase in complexity of these new architectures, special attention has to be paid in order that DMS maintain a high level of efficiency, while preserving the same level of functionality. The *Client-Server-Monitor* is aimed to help engineers and scientists to move from traditional methods of programming for sequential machines using traditional DMS to the new generation of MPP computers, by offering a similar programming environment.

The Data Management System

Data management systems have become a common tool for handling data during large calculations. Such systems are usually a collection of tools for data and memory manipulation. They provide support for self-descriptive and context-free data manipulations as well as for memory and memory-to-disk operations. The benefits offered are multiple:

- A *logical organization* of data. Data structures and formats are handled by means of a high-level language interface. The self-descriptive data representation entails high visibility of data structures for both the application programmer and the user. It allows also symbolic access and manipulation of data which facilitates code development and testing.

- A *physical organization* of data. Fast access to large amount of data stored in memory or on disk is usually critical for large computer application. Portability of both binary data bases and codes is also a important issue for multi-platform development. DMS have to provide archives handling and to facilitate the access of stored data by post-processing tools.

  Within the *Client-Server-Monitor* computational environment, the MEM-COM

Data Base is used as DMS. MEM-COM is based on a modular approach which allows flexible development of code by means of separate modules. Compatibility between modules is imposed by a standard interface with the data base. Figure 1 presents a schematic view of the relation between the data base and the computational modules.

While data bases provide a high level of flexibility, they have often made their use daunting for users not accustomed to their complexity. In order to gain simplicity, the MEM-COM library provides standard functions in C and FORTRAN 77 which can be accessed as subroutines or functions. They combine low-level system calls and eliminate unnecessary internal memory management. Below, a short FORTRAN 77 example is shown of how a data base (called "demo.db") is opened and a set of coordinates (set number 1) is copied into an allocated memory region defined by the identificator "idcoor":

```
c Open data base file "demo.db"
      call opendb(1, 'demo.db', 'old', istatus)
c Load set number one in single precision
      call dmmget(1, 'COOR.1', 0, nword, 'E', idcoor, istatus)
```

Computer calculations are usually done in three successive steps (see Figure 1): The *pre-processing step* defines, generates, or modifies the basic elements of the calculation (e.g. geometry, mesh, etc.), the *calculation* itself where large memory and computational power are required and finally, the *post-processing* step during which output data are analyzed and archived. In the case of a distributed memory system, these different steps can take place on different computers: *Pre-* and *post-processing* are usually done on a workstation, while the *computation* itself is performed on a MPP system. Any DMS faces therefore several problems: Compatibility between data representation, spread of data among various computers or PEs, and synchronous or asynchronous accesses to and from the data base.

The Client-Server-Monitor programming environment

In order to cope with these difficulties, the *Client-Server* adopts a relatively simple approach based on the "master-slave" programming model. In such a para-

digm, the *Client* programs are controlled by a *Server* which provides support for the basic administrative tasks. In addition, a few processing elements can be dedicated for very specific purposes such as data base handling (*Data Base Server*), on-line monitoring and process control (*Monitor*). The addition of a *Monitor* module to the *Client-Server* concept is aimed at providing a graphical user-interface tool which can be used for program development and debugging and also for monitoring the time evolution of data during computation. Figure 2 shows the various components of the *Client-Server-Monitor*. In this environment one can distinguish the following elements:

The *Server* acts as a program master. It controls the initialization and starts up requested tasks (*Client* programs, *Monitor* and/or *Data Base Server*) either on a local PE or on a remote host or PE. The *Server* also manages the various processes and cleans up all tasks at the program completion. Within the *Server* module, events are controlled by an "event loop" made of blocking reads:

```
while (1) {
   read event
   case event from Client
   case event from Data Base Server
   case event from Monitor
}
```

Note that signals cannot be used to handle events, since they are inherently recursive. Libraries such as PVM or Motif do not allow for recursive coding.

Client programs are written by the user and they consist of either a single program written in the SPMD (Single Program, Multiple Data) programming style or a collection of heterogeneous programs. Request made for writing or reading data or for on-line diagnostics will activate the *Data Base Server* or the *Monitor* respectively. Explicit message passing is enforced between the various *Client* programs by the use of simplified communication calls which allow transparent data transfer between hosts on which data representation may differ.

The *Monitor* is a dynamic module which can display 2D and 3D diagrams for on-line monitoring as shown in Figure 3. The X11-display of the *Monitor* is based on the XGrafix library which offers primitives for the drawing of windows, curves, vectors and 3D plots (each diagram has a multi-curves capability). The

Monitor module has been written in C++ to allow a more flexible management of data. A specific *Client* has to be chosen to communicate data to the *Monitor*. This node is responsible (1) to initialize the diagrams (FORTRAN 77 example):

```
if (mype.eq.0) then
   call MON_start(istatus)
   call MON_diagram(1, 'linlin', 'Xaxis', 'Yaxis', ...)
   call MON_curve(1, 1, 'hist2d', npoints, color1)
   call MON_curve(1, 2, 'curve2d, mpoints, color2)
endif
```

(2) to collect the data spread among the clients at a pre-defined frequency and to send these updated values to the *Monitor*:

```
if (mod(nstep,10) .and. mype.eq.0) then
   call collectData(x, y, n, z, w, m)
   call MON_updcuve(1, 1, x, y, n, time, istatus)
   call MON_updcuve(1, 2, z, w, m, time, istatus)
endif
```

Layout parameters of the diagrams can be changed at run time by means of menus. A panel is also displayed (see Figure 3) which provides interaction with the running application. At the present stage, the panel offers the possibilities to modify the progress of the calculation (*Run*, *Step* and *Stop*), to dump pre-defined variables in the data base (*Save*) or to disable temporarily the display (*Wait*) without interrupting the calculation. Finally, the *Quit* option prompts a graceful stop of the program.

The *Data Base Server* is an optional module started only if a request is made to access a data base. The data base access is controlled in a sequential manner. Data bases are mainly used to store initial input data, to archive intermediate data which can be used to restart a calculation and to store final results.

Communications between the various actors are made by means of the PVM (Parallel Virtual Machine) explicit message passing library [3]. This library has been used for the development of this environment since it provides support for a large variety of different computer architectures and is easily implemented in an application. However, it suffers from drawbacks like multiple buffering of

data, the presence of numerous daemons and a rudimentary handling of groups.

Performance measurements

I/O performance measurements depend heavily on the hardware configuration, such as CPU clock speed, CPU work load, memory bandwidth, I/O controller configuration, and network bandwidth. Since the PVM library is used to transfer data within the *Client-Server*, an overhead is to be expected when performing I/O operations via the *Data Base Server* compared to the non-distributed version of the MEM-COM DMS (stand-alone version), where I/O operations are directly performed to the disk. In order to measure this communication overhead, tests have been made to compare results obtained with an I/O test program for three different configurations: The stand-alone version of the MEM-COM DMS, a local version of *Client -Server*, where the *Client* program (i.e. the I/O test program) and the *Server* run on the same computer, and a network version of the *Client-Server* where the *Client* program runs on a remote platform (see Figure 4).

The I/O test program consists of write and read operations on a single data base file. A constant number of data sets of size ranging from 1 to 1000 integers, are first written and then read. Tests have been run on 133Mhz workstations connected by a 10Mbit/sec Ethernet cable. Results displayed in Figure 5 show the elapsed time required to transfer data sets for the local and network version of the *Client-Server*, normalized to the elapsed time taken by the stand-alone MEM-COM DMS. It can be observed that, compared to the stand-alone version, the overhead for writing and reading data sets is approximately a factor of 8, respectively 11, if the *Server* runs on the same platform as the I/O test *Client* program (local version) and a factor of about 13, respectively 17, if the *Server* runs on another platform (network version). The normalized elapsed time measured is not influenced by the size of the sets transferred. Although PVM allows the modification of the size of its internal data buffers, it has been found rather difficult to optimize this parameter for cases where set size varies significantly. It should also be stressed that, although the communication overhead measured in this test is mainly due to the message passing library, results presented in this section should not be considered as a benchmark of PVM. They are only given as an indication of the overhead one has to pay when using a distributed DMS.

Nevertheless, the results show an important overhead in communication, which is mainly due to multiple data conversions and superfluous buffering of data by PVM. To remedy this situation, a new version of the *Client-Server-Monitor* is presently under development, which uses sockets to allow a more flexible management of communications and to avoid the cumbersome presence of multiple daemons.

Similar I/O tests have been performed on a Cray T3D. *Client* programs are running on nodes of the T3D and the *Server* module is on the Cray YMP front-end computer. Unfortunately, data transfer between the T3D and the front-end computer is difficult to measure since it depends critically on the load of the front-end machine as well as on the system configuration, both of which are outside of the control of the user. Measurements may therefore strongly vary depending on the time at which the tests are made. To cope with this situation, a serie of tests has been performed to evaluate the dispersion in measured elapsed time for such a hardware configuration.

Tests have been made with 1, 8, and 32 *Client* programs (PEs). Each node transfers 100 data sets of size ranging from 1 to 1000 integers. Figure 6 presents average elapsed time for writing and reading data sets. The measured dispersion is represented by the vertical bars. Results show a dispersion up to 70% of the average elapsed times due to the load of the front-end computer. The elapsed time has also been found to increase proportionally to the number of *Client* programs. This is due to the fact that parallel I/O requests made simultaneously by multiple PEs are serialized by the *Server* event handler. Data base access time increases for large data sets. For this test case, it reaches about 0.3 Mbytes/sec for large sets with a fully loaded front-end computer (based on the elapsed time and not the CPU time). These results should not be taken as benchmark of the distributed PVM bandwidth on Cray supercomputers (see Ref. [6] for performance overview), since other effects than I/O transfer rates dominate. These effects include effective data base operations, serialization of the event handler, and PVM daemons operations among others. In order to suppress the *Server* event handler bottleneck, parallelisation of the event handler by multi-threading the *Server* will be investigated in the future. This will, however, substantially increase the complexity of the *Data Base Server* due to the treatment of potential conflicts arising from data dependencies. The additional run-time overhead

implied by the use of threads should however be small compared to the other UNIX processes involved in the environment such as the sockets, since threads do not require the allocation of a separate address space and are much cheaper in terms of creation and context switching.

Experiences with a PIC simulation on a CRAY T3D

The first program that benefited from the *Client-Server-Monitor* was an electrostatic particle-in-cell (PIC) simulation developed to study beam instabilities in gyrotrons [4]. Originally written for a serial supercomputer (a Cray YMP) and using MEM-COM as DMS, this program has been modified for the Cray T3D architecture. The pre-processing stage is done in three steps corresponding to different modules: (1) Generation of the computational mesh, (2) initialization of data and boundary conditions and (3) domain decomposition [5]. The size of the input data base created for this problem is a few Kbytes. Geometrical quantities (e.g. coordinates, connectivity), physical quantities and boundary conditions for each PE are stored as data sets in the data base. The input data base contains also some relational tables with physical or numerical parameters relevant for the calculation. During the computation pre-defined sets of physical values are stored periodically in order to enable restart of the calculation. Results are stored for post-processing analysis at the end of the calculation.

With the hardware configuration presented above, measurements have shown that under normal daytime conditions, transmission of data can reach at best up to 1MBytes/sec for large data sets (a few Mbytes). Since the computation time between two data transfers is large compared to communication time, the communication overhead is small, except for very large number of PEs where the serialization of the event handler slows down the access to the data base. It is clear that the use of the *Monitor* decreases the speed of the calculation since interruptions may be requested by the user and additional data have to be transferred to the *Monitor* module. It should be noted that, in general, the CPU time spent to display the graphics is negligible compared to the transfer time (done via Ethernet). It has also been observed that the serialization of the event handler serialization speed may become a critical issue for a large number of *Client* programs.

In this *Client-Server-Monitor* environment the only overhead is due to communication, i.e. there is no additional run-time if no requests are made to the *Server*. This concept has the great advantage over a file server type of DMS that it only transmits those data that are really requested. I/O and paging are restricted to the *Server* process, and not the *Client* process. Concurrent access control are also limited to the *Server* process.

Conclusion

A *Client-Server-Monitor* computational environment has been presented. The environment, composed of several modules, provides tools for data management and on-line interactive monitoring of running parallel applications. A distributed version of the MEM-COM Data Base is used as data management system and communication within the *Client-Server-Monitor* environment is presently done using PVM. Intensive numerical simulations in the field of plasma physics have demonstrated the usefulness of such an environment for parallel computation. Future development will include the use sockets and concurrent access to the data base in order to increase the capabilities of this environment.

Acknowledgments

This development has been made in the framework of the Cray-EPFL Parallel Application Technology Project. The authors would like to thank K. Appert and O. Sauter for fruitful discussions. F. Lapique is also acknowledged for his help and many useful discussions.

References

[1] V. Vahedi, J.P. Verboncoeur and P. Mirrashidi. *XGrafix library, version 1.1.* University of California, Berkeley, 1993.

[2] *MEM-COM, an integrated memory and data management system. Reference Manual, Version 6.1.* SMR Corporation, June 1994.

[3] G.A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek and V. Sunderam. *PVM 3 User's Guide and Reference Manual.* Oak Ridge National Laboratory report, ORNL/TM-12187, September 1994.

[4] T.M. Tran, G. Jost, K. Appert, O. Sauter and S. Wüthrich. *Particle-In-Cell (PIC) simulations of beam instabilities in gyrotron beam tunnels.* Proc. 20th International Conference on Infrared and Millimeter Waves, Orlando, December 1995, 124.

[5] T.M. Tran, R. Gruber, K. Appert and S. Wüthrich. *A direct Poisson solver for Particle-In-Cell simulation.* Accepted for publication by Computational Physics Communication.

[6] *PVM and HeNCE Programmer's Manual.* SR-2501 4.0, Cray Research Inc., 1994, 31.

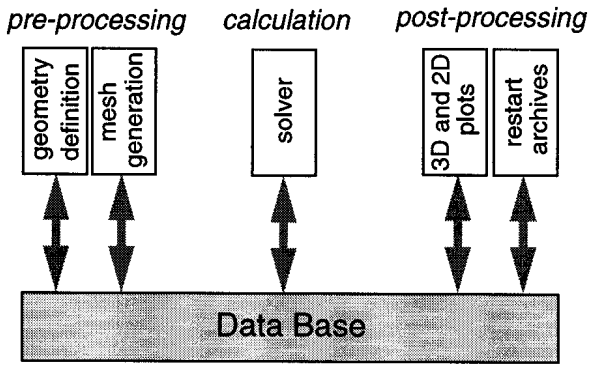

Figure 1 : Relation between the data base and the computational modules.

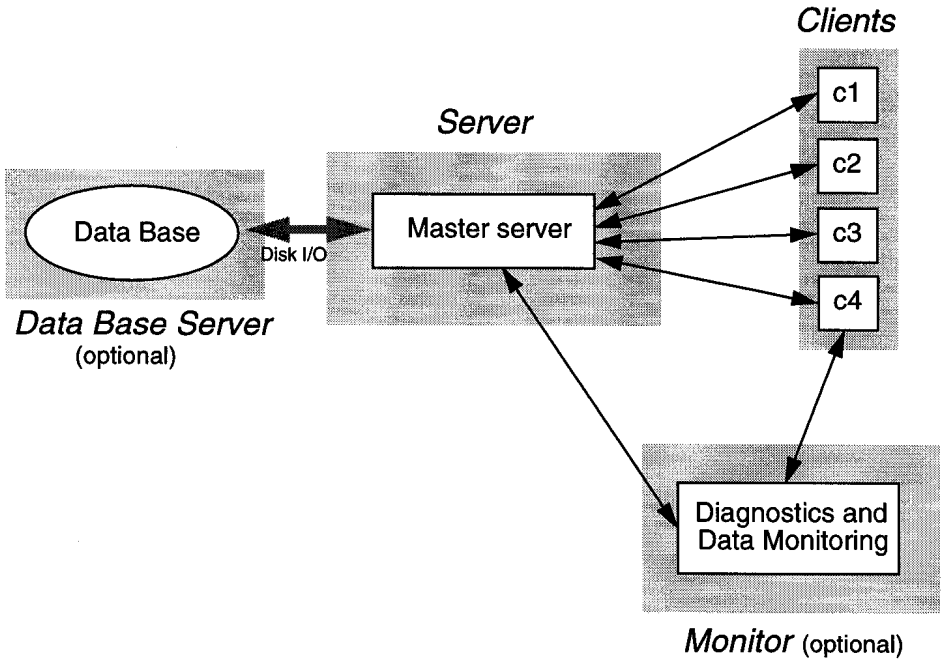

Figure 2 : Schematic set-up of the *Client-Server-Monitor.*

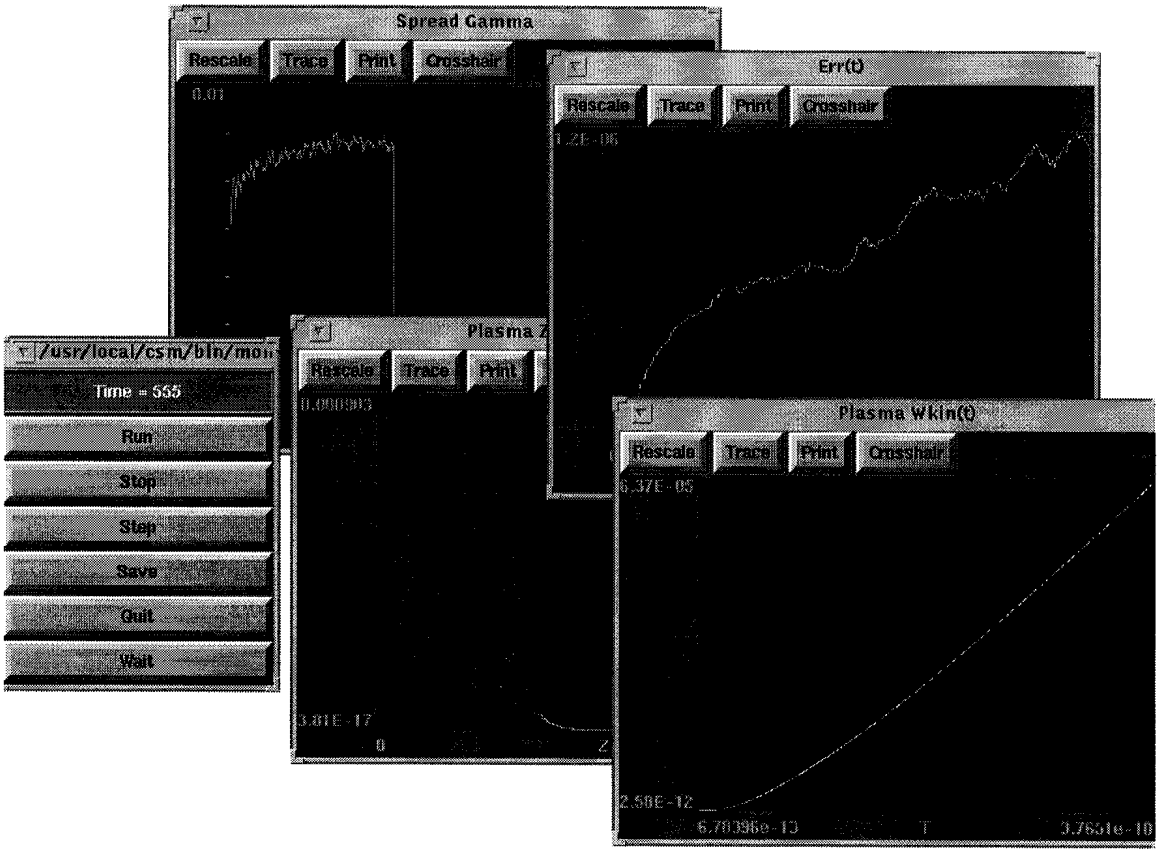

Figure 3 : Example of on-line monitoring of an application.

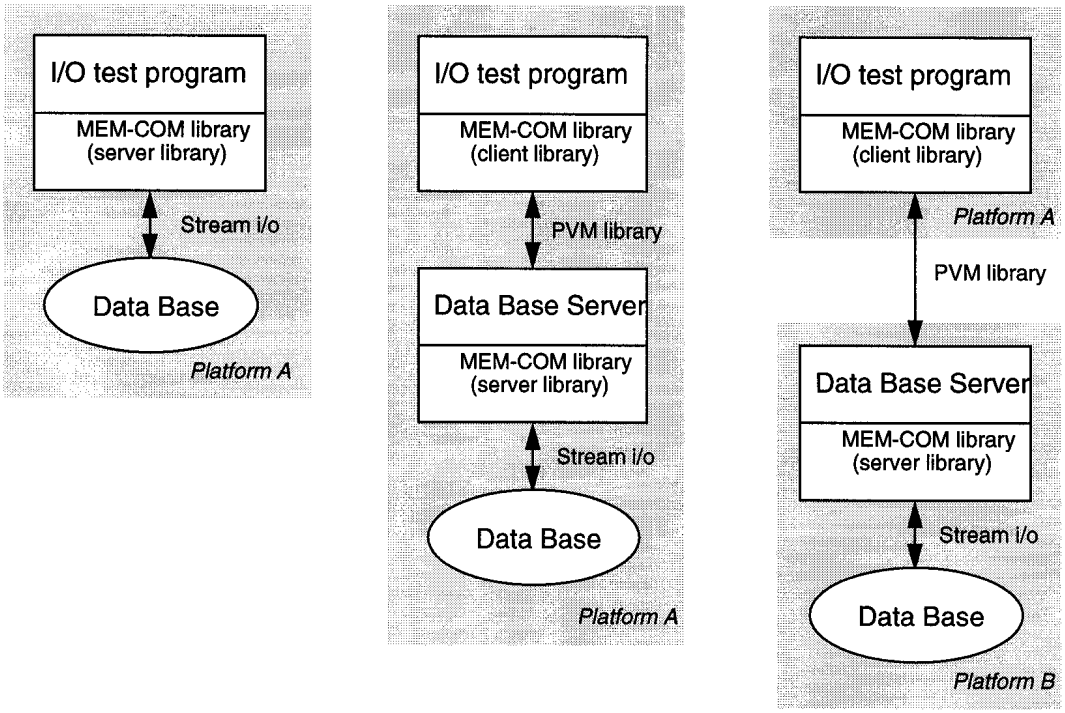

Figure 4 :I/O test program configuration. Stand-alone version (left), local (middle) and network (right) *Client-Server* versions.

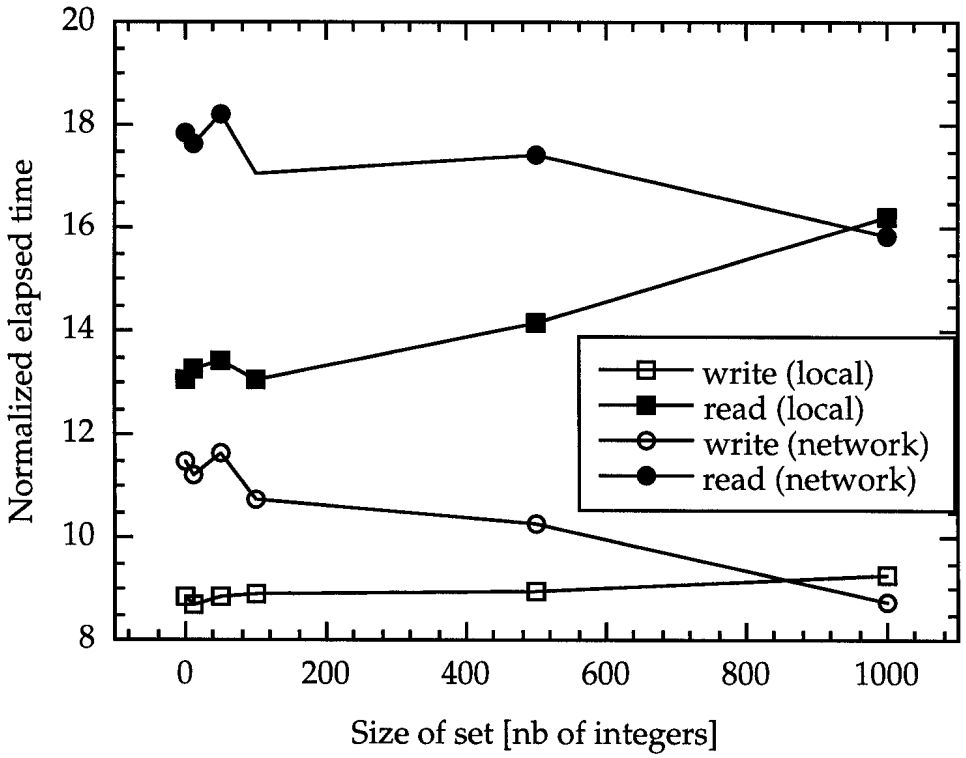

Figure 5 : Normalized elapsed time for writing and reading data sets with the local and network versions of the *Client-Server.*

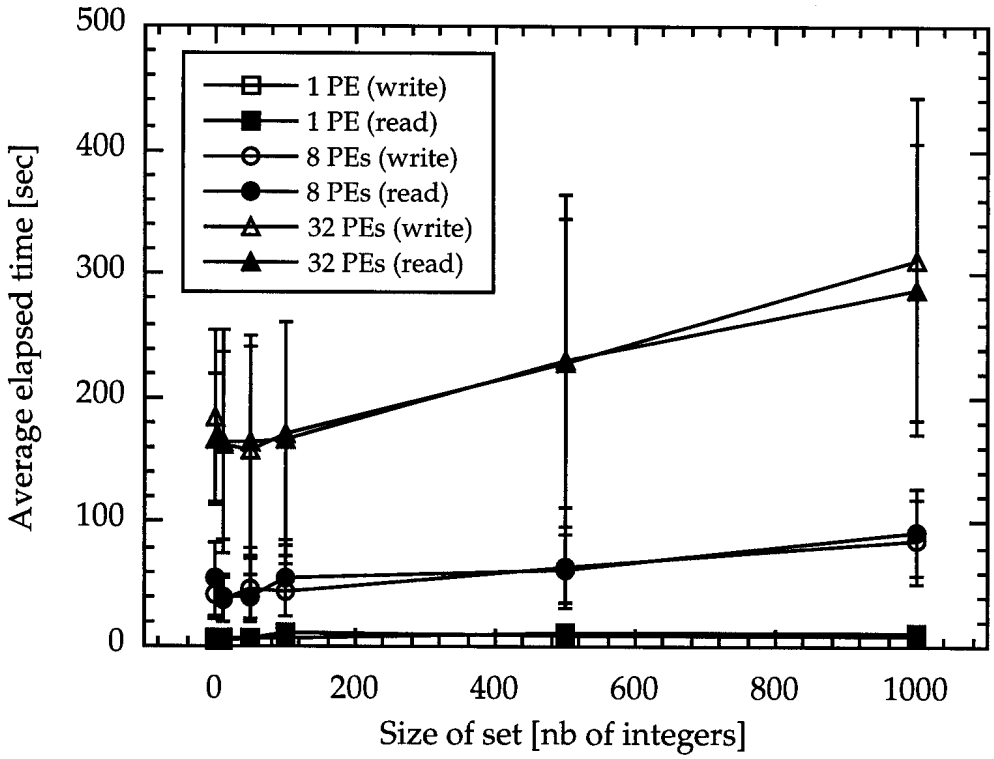

Figure 6 : Average elapsed time for 1, 8, and 32 nodes of the Cray T3D to write and read 100 data sets on a data base on the front-end Cray YMP.