DOSMOS : A Distributed Shared Memory based on P.V.M.

L. Brunie and L. Lefèvre
Laboratoire de l'Informatique du Parallelisme
Ecole Normale Superieure de Lyon
69364 LYON Cedex 07
France
(lbrunie, llefevre)@lip.ens-lyon.fr

September 29, 1994

Abstract

The purpose of this paper is to describe a new Distributed Shared Memory model, called DOSMOS, and to present its implementation based on PVM and built on a structural approach of parallel programming. Moreover DOSMOS has been specially designed in order to provide a user-friendly programming environnement.

1 Introduction

If distributed memory MIMD machines allow very high performances, their programming remains "esoteric" for most of end-users accustomed to classical monoprocessor programming. Oppositely, shared memory parallel computers are easier to program but badly adapted to applications generating many memory accesses.
The purpose of Distributed Shared Memory systems (DSM) is to implement, above a distributed memory architecture, a programming model allowing a transparent manipulation of shared data. In practice, a DSM system has commonly to handle all the communications and to maintain data coherence.
Though various DSM systems have been proposed, few of them allow to efficiently and scalably deal with massively parallel multi-processors.
The aim of this paper is to describe a new distributed shared memory model, called DOSMOS: Distributed Objects Shared MemOry System. Initially this system is designed to deal with massively parallel machines. But to test its availability and to improve it in an easy and portable we first implement with PVM [GBD+93]. By this way DOSMOS also provide a a user-friendly programming environnement with shared objects adapted to a network of workstations.
This model allows processes to share distributed objects. A hierarchical organization of the processes (into groups and sub-groups based on the shared data they address) permits a high degree of scalability. This organization can be either defined using compilation directives or dynamically generated at run-time.

This paper is divided into two main parts. In the first part, we will discuss about the model of our distributed-shared memory system: groups generation and consistency. In the second one, we will see the implementation of DOSMOS realized on top of PVM for a network of workstations.

2 Purpose of D.S.M.

MIMD[1] parallel machines can be divided in two class according to memory architecture : shared memory machines are composed of a set of processors accessing to a single memory. The main advantage of these machines lies in their easy programming due to the fact that there are no communications between processors. However, contentions and collisions prevent applications to be actually scalable and most of shared-memory machines are limited to a small number of processors. Distributed memory systems (like a network of workstations), on the contrary, can integrate a large number of processors. The communication model usually used is message-passing, which allows important speedups. However its programming remains difficult since end-users have to take into charge all communication operations.

In that context, by allowing the programmer to share "memory objects" without being in charge of their management, Distributed-Shared Memory Systems want to propose a trade-off between the easy-programming of shared memory machines and the efficiency and scalability of distributed memory systems.

A distributed shared memory is a mechanism allowing end-users processes to access shared data without taking into charge inter-process communications. In other words, the goal of a DSM system is to make inter-process communications transparent to end-users. Both hardware and software implementations have been proposed in the literature. From a programming point of view, two approaches have been proposed.

- Shared virtual memory systems (like KOAN [LP92] and Mermera [HS92]) are very similar to the well-known concept of paged virtual memory implemented on mono-processor systems : all distributed memories are virtually merged together into a single wide address space.
- Distributed shared memory systems (like Linda [ACG86] or ORCA [TKB92]) share objects i.e. variables with access functions. The user has only to define which data (objects) are to be shared. The whole management of the shared objects (creation, access, modification) is assumed by the DSM system.

Contrary to SVM systems which work at operating system layer, objects DSM systems actually propose a programming model alternative to the classical message-passing.

In both cases, implementing a DSM system implies to address problems of data location, data access, sharing and locking of data, memory coherence.

3 DOSMOS Model

Our model is based on a set of passive objects distributed among the processes of the application. These objects may be shared in a transparent way by these processes. The user has only to declare his objects and the read/write operations realized on these variables. The effective location of his data is hidden to the user. Our system manages the mapping of the objects and their accesses with a classical client-server protocol. For the moment, our system is based on a one writer and many readers model. By this way we have a lot of copies of the same object in different processes. We use a weak consistency to assume the coherence of objects copies in our system, but if the user needs it, it's possible to declare objects with a strong coherence protocol (management of synchronization barriers).

As the links between processors are physically the same way of communication (Ethernet), we introduce the notion of groups in our system. These groups virtually merge together processes which globally share the same set of objects. This hierarchy allows to avoid global communications,

[1]MIMD: multiple instructions, multiple data streams, involves multiple processors each of them autonomously executing its own instructions on its own data.

between all the processors, required to maintain objects coherence. By this way, we propose a structural approach of parallel programming (with a hierarchical merge). This rich model permits to take into account of a wide number of architectures and complex and diversified applications (for example: databases (each group has a dedicated database), imagery (scene splitting)...).

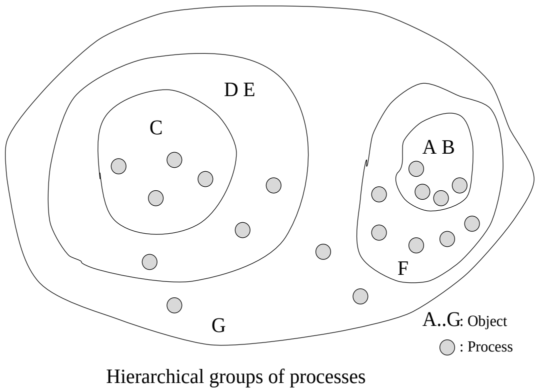

Figure 1: Groups

A group can be created at compilation-time by the user if he knows which processes of the application share the same set of objects. When a process wants to access an objects shared by an other group, it can do an occassional access (see Chapter 4.2). The copy, returned by this request, is correct at a given instant but the coherence of this copy will not be maintain by DOSMOS. For further information on DOSMOS model see [BL94].

4 Implementation

The implementation of DOSMOS is based on two different layers (see Figure 2):

- **Preprocessing level**: It treats and analyses the user's application to detect and generate accesses (Put/Get) to shared objects. This phase is useful to assume the transparence of our system.
- **DSM level**: It assumes the creation of objects, groups and processes and the distribution of the application among the processor with classical PVM functions.

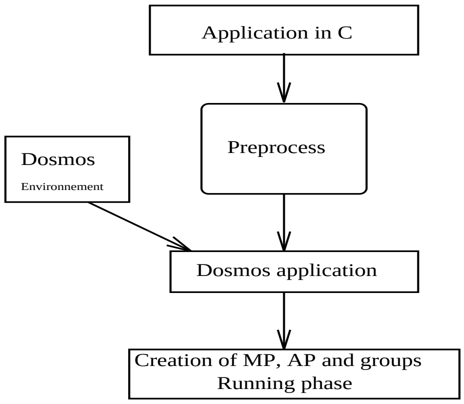

Figure 2: Dosmos Environment

4.1 Memory and application process

Our system is componed by two main class of process (see Figure 3):

- **Application process (A.P.)**: it contains and executes the code of user's application
- **Memory process (M.P.)**: it manages all shared objects accesses required by the process application

These two kind of processes are generated by a classical master process and can be mapped on different workstations or can run together on the same processor. Each application process only deals with its dedicated memory process to access to shared objects. Moreover a memory process can manage the shared variables of different application processes, if the amount of communications is not too heavy between these APs and the other processes. If an AP wants to access to a shared object not present in its local memory, it only sends the request to its dedicated memory process. Then the MP takes into charge all the communications required to access the shared object.

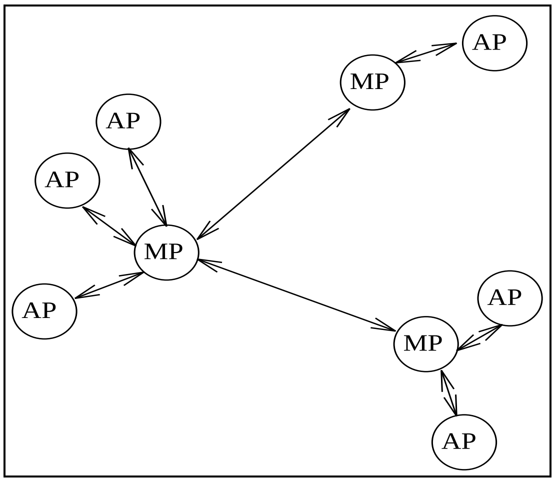

Figure 3: Processes in DOSMOS

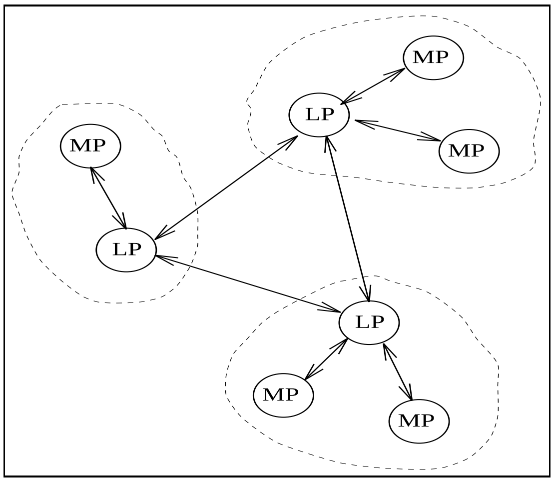

Figure 4: Groups and link process

4.2 Groups

In each group, a memory process plays the role of a link process (LP) (see Fig 4). It is dedicated to the communication between groups. When a memory process attempts to access to an object belonging a different group, it sends its request to the LP. This LP will communicate with the others LP of the virtual architecture to find the requested object. The LP is the only entry point in the groups (special data are managed by these LP to know the name of objects shared by each group).

5 Conclusion and future work

We are currently doing experimentations to compare applications with our prototype to applications using classical message-passing communications between processors. The first results, on small applications of matricial computing, show little difference of execution time. Usually the application using DOSMOS takes more time than the same application with message-passing communications (1.4 time more). But the applications using DOSMOS are much easy to write because all communications and data locations are hidden to the user. By the way, these applications are portable (they look like sequential version) and more reliable because the system manages all the communications. PVM has permitted an easy implementation of our system on network of stations. But in the future we want to take advantage of its portability, to install DOSMOS on a massively parallel machine, in order to validate the scalability of the approach.

References

[ACG86] S. Ahuka, N. Carriero, and D. Glernter. LINDA and friends. *IEEE Computer*, 19(8):26–34, August 1986.

[BL94] L. Brunie and L. Lefèvre. Modèle de mémoire distribuée-partagée pour machine massivement parallèle. In *RenPar'6*, Ecole normale Supérieure de Lyon, France, June 1994.

[GBD+93] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM3 User's Guide and Reference Manual.* Oak Ridge National Laboratory, Oak Ridge, Tennesse, May 1993.

[HS92] Abdelsalam Heddaya and Himanshu Sinha. An overview of mermera: A system and formalism for non-coherent distributed parallel memory. Technical report, Computer Science Department, Boston University Boston, MA 02215, September 1992.

[LP92] Zakaria Lahjomri and Thierry Priol. Koan : a shared virtual memory for the ipsc/2 hypercube. In Springer-Verlay, editor, *Parallel Processing : CONPAR 92-VAPV*, pages 441–452, September 1992.

[TKB92] Andrew S. Tanenbaum, M. Frans Kaashoek, and Henri E. Bal. Parallel programming using shared objects and broadcasting. *IEEE computer*, 25(8):10–19, August 1992.