A DSM-based structural programming environment for distributed and parallel processing

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

Abstract

This paper describes an original programming environment based on the DOSMOS[1] system. Its implementation is built on a structural approach of parallel programming. By combining this structural model with weak consistencies protocols, we improve the performances of DSM system and provide a programming model which mixes message-passing and virtually shared objects. Morevoer by integrating a development platform consisting of processes mapping, shared objects management, pre-processing step and monitoring facilities, the DOSMOS system has been designed to provide a performant user-friendly programming environment.

1. Introduction

Distributed Shared Memory systems (DSM) have been designed to implement, above a distributed memory architecture, a programming model allowing a transparent manipulation of virtually shared data. Thus, in practice, a DSM system has to handle all communications and to maintain coherence of the shared data.

This paper describes an original programming environment, called DOSMOS[1]. This system is based on a structural approach of parallel programming. In other words, DOSMOS proposes to the user to hierarchically structure processes into *groups* and *sub-groups of processes* sharing a same set of variables. This feature, combined with weak consistency protocols allows to reduce the amount of communications required for the management of the shared data, and, as a consequence, to ensure efficiency and scalability to the applications.

Moreover, to be used by various kind of programmers, DOSMOS allows to mix two programming models : message-passing (PVM [GBD+93]) and DSM (DOSMOS) code in a same application.

To provide an useful development platform above DOSMOS, we have added a set of tools to design a complete programming environment. This Tool Kit allows the designing and execution of DOSMOS applications in terms of processes mapping (DOSMOS-Map tool), shared objects management and pre-processing step. To complete the programming environment, DOSMOS integrates a devoted monitoring tool (called DOSMOS-Trace) which has been added to the system to clearly understand and analyse the behavior of DOSMOS applications. At last, this programming environment has been designed to run both on distributed systems and on parallel machines. Thus, to ensure the portability of both the system and the applications, DOSMOS has been developed on top of PVM.

This paper is divided into three parts. After a short description of previous works in terms of DSM systems (section 2), we describe the basics of DOSMOS DSM system in terms of coherence protocols, shared objects, programming model and process structuring (section 3). Then a description of the DSM-based programming environment is proposed. At last, section 5 proposes a discussion both on the basic features of this programming environment and on implementation choices and points out future developments.

2. Purpose of Distributed Shared Memory systems and previous works

By allowing the programmer to share "memory objects" (i.e. programming variables) in a transparent way, Distributed Shared Memory Systems (DSM) propose a interesting trade-off between the easy-programming of shared memory machines and the efficiency and scalability of distributed memory systems. Basically, a Distributed Shared Memory system is a mechanism that allows application processes to access to shared data in a transparent way. In other

[1] DOSMOS : Distributed Objects Shared MemOry System

words, a DSM system releases the programmer from the management of all inter-process communications.

Both hardware and software implementations have been proposed. The main systems require to implement an additional software layer :

Virtual Shared Memory systems (VSM) allow to share pages of data, i.e. to merge into a single wide address space a set of memory pages distributed in the network. First systems like IVY [Li88] or KOAN [LP92] were dedicated to specific parallel machines. Other systems like MIRAGE [FP89], MUNGI [HERV93] or MUNIN [CBZ91] have to deal with specific problems of operating systems.

Object-based Distributed Shared Memory systems (DSM) work at the program level, i.e. they implement a software layer that automatically generates, on the user's behalf, all the communications required to manipulate shared data. In other words, instead of defining (and writing in the code) the inter-process communications, the programmer only specifies which data are actually shared. Then he can use these data as if they were local. On its side, the DSM system takes into charge all the communications necessary (as a message-passing programmer would do). Such DSM systems like ORCA[TKB92] or CLOUDS system [RAK89] have been implemented on parallel machines. The DOSMOS[BL94, BL96] system belongs to this class of systems.

3. Basics of DOSMOS

DOSMOS is an object-based DSM system (cf section 2), i.e. it allows processes to share in a transparent way a set of passive objects (i.e. of programming variables) distributed in the network.

However, DOSMOS integrates novel features :

DOSMOS Processes : Basically, a DOSMOS application is composed of three kind of processes :

- **Application processes (A.P.)** contain and execute the code (written in C) of the application ;
- **Memory processes (M.P.)** manage the whole DSM system, i.e. they provide A.P. with the objects they request and maintain the data coherence. Each A.P. is connected to one and only one memory process. On the contrary, an M.P. can be connected to several A.P. and several M.P.
- **Link processes (L.P.)** which are used for management of gropu of application processes. They allow to do extra-group accesses.
- **Monitoring Processes (M.P.)** eventually used to collect and to analyse traces of the execution.

Array allocation : DOSMOS allows to manipulate both basic type variables (integer, float, char. . .) and distributed arrays. These arrays are split into several "system objects", distributed in the network. Various splittings are provided : by row, by column, by block and by cyclic block. The system ensures a transparent access to arrays, whatever the splitting implemented.

Weak consistency protocols : for efficiency and scalability purpose, DOSMOS allows to duplicate shared objects. It is clear that these replicas have to be kept coherent. Most of actually implemented models are strong consistency oriented. DOSMOS implements a weak protocol : the release consistency. This model [GLL+90] provides two synchronization operators : *acquire* and *release*. These operators allow processes which want to modify shared objects to lock and unlock them (in other words, these routines actually implement a mutual exclusion on the accesses to the shared objects).

Hierarchical structuring of the application processes : Previous DSM systems have always proposed "*flat*" models in which any shared object is accessible from any process. Such "anarchical" models cannot be scalable. In DOSMOS, processes can be group into groups in sub-groups in order to optimize the management of the coherence of data.

When one observes the behaviour of a DSM application, and more particularly the behaviour of a process participating to the application, it appears that if some shared data are intensively accessed by this process, some others are either very not often accessed or never accessed. This leads us to introduce some definitions (see fig. 1) :

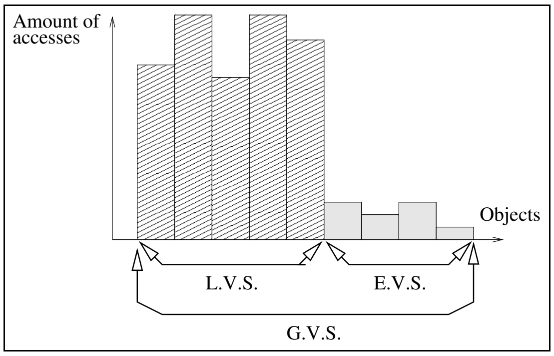

Figure 1. Example of accesses distribution

- *G.V.S. : The **Global Virtual Space** (GVS) of a process is the set of the shared objects accessed (in read or write mode) by a process during the execution of the application.*
- *L.V.S. : The **Local Virtual Space** (LVS) of a process is the set of the shared objects intensively accessed by this latter.*
- *E.V.S. : The **Extern Virtual Space** (EVS) of a process is the set of the shared objects rarely accessed by the process.*

Let P a process. We have :

$$G.V.S.(P) \; = \; L.V.S.(P) \; + \; E.V.S.(P)$$

Usually, in previous systems, when an object O is modified, an invalidation message is sent to all the processes P such that $O \in GVS(P)$. This prevents, as noted before, to ensure a good scalability. By using a hierarchical grouping of processes, DOSMOS limits the invalidation messages to processes such that $O \in LVS(P)$.

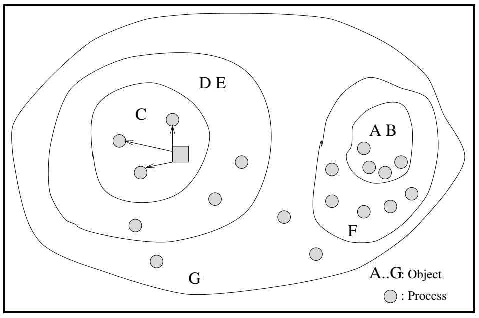

Figure 2. Hierarchical grouping of processes

Basically, DOSMOS proposes to structure the application into hierarchical groups of processes sharing the same objects (Figure 2). In practise, a group is defined by a set of processes and a set of shared objects. Processes of a same group share all the objects attached to the group, i.e. if they request an object, they will receive a copy of this object which will be automatically updated by the system.

But DOSMOS also allows processes to access to extra-group shared objects. For this purpose, in each group, a dedicated memory process, called Link Process (LP), plays the role of link between groups (see Fig 3). Thus, these special MPs takes into charge all the communications between groups. This model presents two important advantages :

- the access to the shared objects is optimized ;
- the maintaining of the consistence is kept cheap.

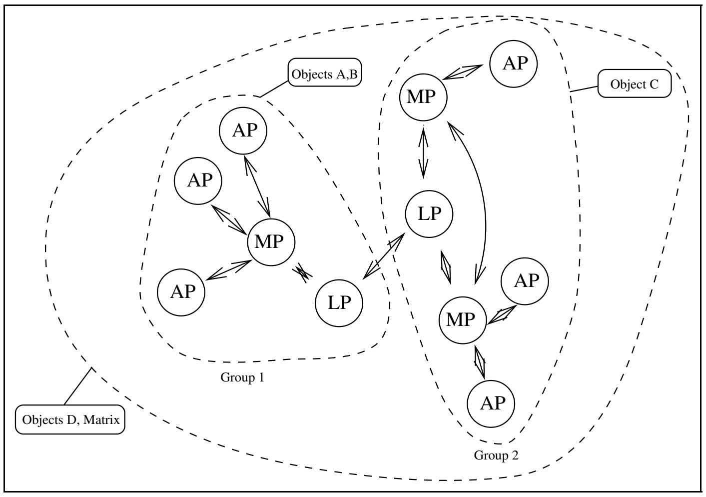

Figure 3. Groups and link processes

Experiments done with DOSMOS system on network of workstations and parallel machines have shown great improvements when using hierarchical groups (see [Lef96]).

Programming model As soon as the *Use_Dosmos()* primitive has been executed, the user can access to the shared objects in a transparent way. However, DOSMOS, as any DSM system, does not pretend to be efficient in all the situations. Consequently, in order to allow the user to optimize specific applications, DOSMOS allows to combine different programming models for user's confort. Consequently, three programming models are available :

- Local programming : in order to minimize the accesses to shared objects, it is sometimes more performant to work on local variables before modifying shared variables.
- D.S.M. programming : the user can use DOSMOS primitives to declare and access to the shared variables in a transparent way.
- Mixing of DSM and message-passing programming : the user can integrate message-passing communications into DOSMOS applications. This feature presents two advantages; First, it permits to deal with specific applications. Second, it allows to port PVM applications on DOSMOS with slight modifications of the code.

4. Programming environment

The implementation of DOSMOS is based on different layers which assume pre-process of the code, management of shared objects, creation of groups and various processes involved in execution of the application (APs, MPs, LPs) and monitoring of the execution (see Figure 4) :

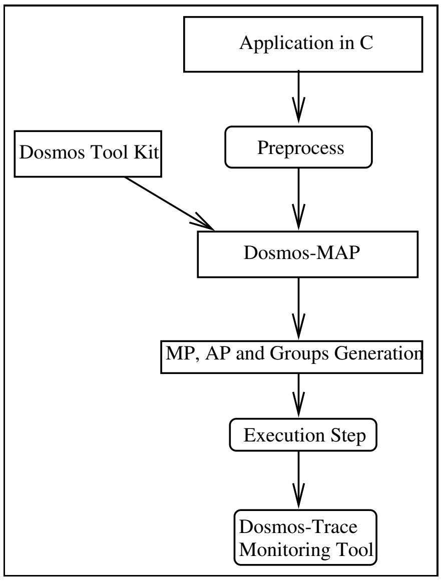

Figure 4. Dosmos Environment

4.1. Pre-processing level

This layer analyses the user's application in order to detect and generate accesses to shared objects. This layer allows the system to be "transparent" by transforming all accesses to shared objects.

4.2. DOSMOS Tool Kit

This set of tools allow to graphically design and execute a DOSMOS application (see Figure 7). Various features are provided to the user :

- **Configuration of Virtual Machine** : due to an underlying PVM layer, the user can mix parallel machines and networks of workstations to execute its application.
- **Processes Mapping** : to choose the ratio of MP and AP and to map them on the processors of the virtual machine.
- **Shared objects management** : to control the splitting and mapping of distributed shared objects.
- **Groups Generation** : this graphic interface allows the user to hierarchically structure its application by grouping the AP.
- **Compilation and Execution** : by allowing distributed compilation and execution on various sites, this tool allows to easily experiment DOSMOS applications on various distributed or parallel systems.

4.3. DOSMOS primitives

By adding only a few new primitives, DOSMOS system stays well-adapted for beginner users. All accesses (except exclusives ones) are totally transparent for the user.

• #include	`Dosmos.h`
• Declaration	`shared` ...
• Begin-End	`use_dosmos()` - `end_dosmos()`
• Exclusive access	`acquire(`*object*`)`, `release(`*object*`)`,
• Synchronization	`sync(`*object or group*`)`

Figure 5. DOSMOS primitives

Use_Dosmos() and *End_Dosmos()* allow the beginning and the end of sharing of the objects. All data accesses are not specified by the user but in order to access exclusively shared objects, two operators are proposed : *Acquire* and *Release*. The barrier routine allows to synchronize all processes sharing a given object or all processes of a given group.

4.4. DOSMOS-Trace monitoring environment

The only way a user can influence the behaviour of his application is the modification of the structure of the shared variables space. So, from the user point of view, monitoring facilities should allow him to precisely know the "activity" of the shared variables (like in figure 6).

The purpose of the DOSMOS-Trace[BLR96] monitoring environment is to provide such information in a scalable and weakly intrusive way. The DOSMOS-Trace tool is based on a set of dedicated processes which collect informations during execution. This data collection is completely transparent for the user. This tool provides several visualizations and informations about the execution like statistics on shared objects, histories. . .

Such diagrams are extremely useful for the user to analyse problematical situations. Indeed they allow to very easily isolate ping-pong effects, over-accessed variables, too important extra-group accesses, bottlenecks, not actually shared variables, etc.

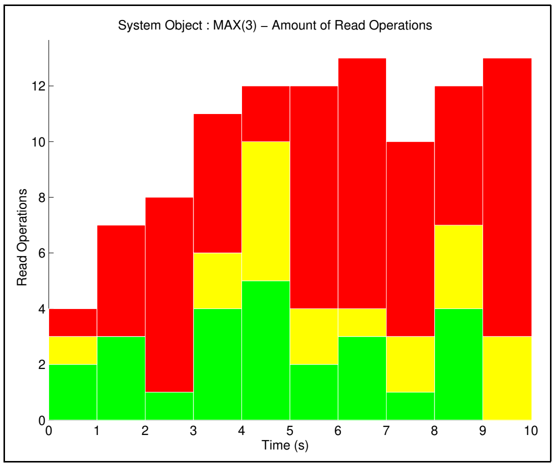

Figure 6. Number and origin of the read accesses performed on an object vs execution time (in black : inter-group accesses)

5. Discussion and future works

This paper has described a novel DSM-based programming environment, the DOSMOS system. In comparison with previous works, this system integrates original functionalities : structuring of the application processes into hierarchical groups, mixing message-passing code and DSM code, weak consistency protocols, designing and execution facilities and monitoring tools.

The whole system has been designed to be as efficient and scalable as possible. Thus the process grouping allows, in conjunction with weak consistency protocols, to reduce the amount of communications required by the management of the DSM system.

Opened to various programming models, designed to be efficient both on parallel machines and distributed systems, DOSMOS provides a portable development platform. Moreover by only adding few new primitives and by providing graphical interfaces to design and to analyse execution application, DOSMOS is an user-friendly programming environment which can easily adapt to a non-expert parallel programming user. Tests have shown the effectiveness of the approach developed in DOSMOS. For more informations about how to use a DSM system like DOSMOS, see [Lef96].

Moreover, we currently continue to improve our programming environment by adding a new distributed tool to DOSMOS which will allow to debug, in a distributed way, the code of DOSMOS applications.

6. Acknowledgments

The authors wish to thank Olivier Reymann, Sebastien Tixier and Jérôme Bolliet for their help.

References

[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.

[BL96] Lionel Brunie and Laurent Lefèvre. New propositions to improve the efficiency and scalability of DSM systems. In IEEE, editor, *1996 IEEE Second International Conference on Algorithms & Architectures for Parallel Processing - ICA3PP '96*, pages 356–364, Singapore, June 1996.

[BLR96] Lionel Brunie, Laurent Lefèvre, and Olivier Reymann. Execution Analysis of DSM Applications: A Distributed and Scalable Approach. In ACM Press, editor, *SPDT'96 : SIGMETRICS Symposium on Parallel and Distributed Tools*, pages 51–60, Philadelphia, Pennsylvania, USA, May 1996. FCRC.

[CBZ91] John B. Carter, John K. Bennet, and Willy Zwaenepoel. Implementation and performance of MUNIN. *ACM - Operating Systems Review*, 25(5):152–164, 1991.

[FP89] Brett D. Fleisch and Gerald J. Popek. Mirage: A coherent distributed shared memory design. In ACM PRESS, editor, *Proceedings of the twelfth ACM Symposium on Operating Systems Principles*, volume 23, pages 211–223, The Wigwam Litchfield Park, Arizona, December 1989.

[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.

[GLL+90] K. Gharachorloo, D. Lenoski, J. Laudon, P. Gibbons, A. Gupta, and J. Hennessy. Memory consistency and event ordering in scalable shared-memory multiprocessors. In *International Symposium on Computer Architecture*, pages 15–26, 1990.

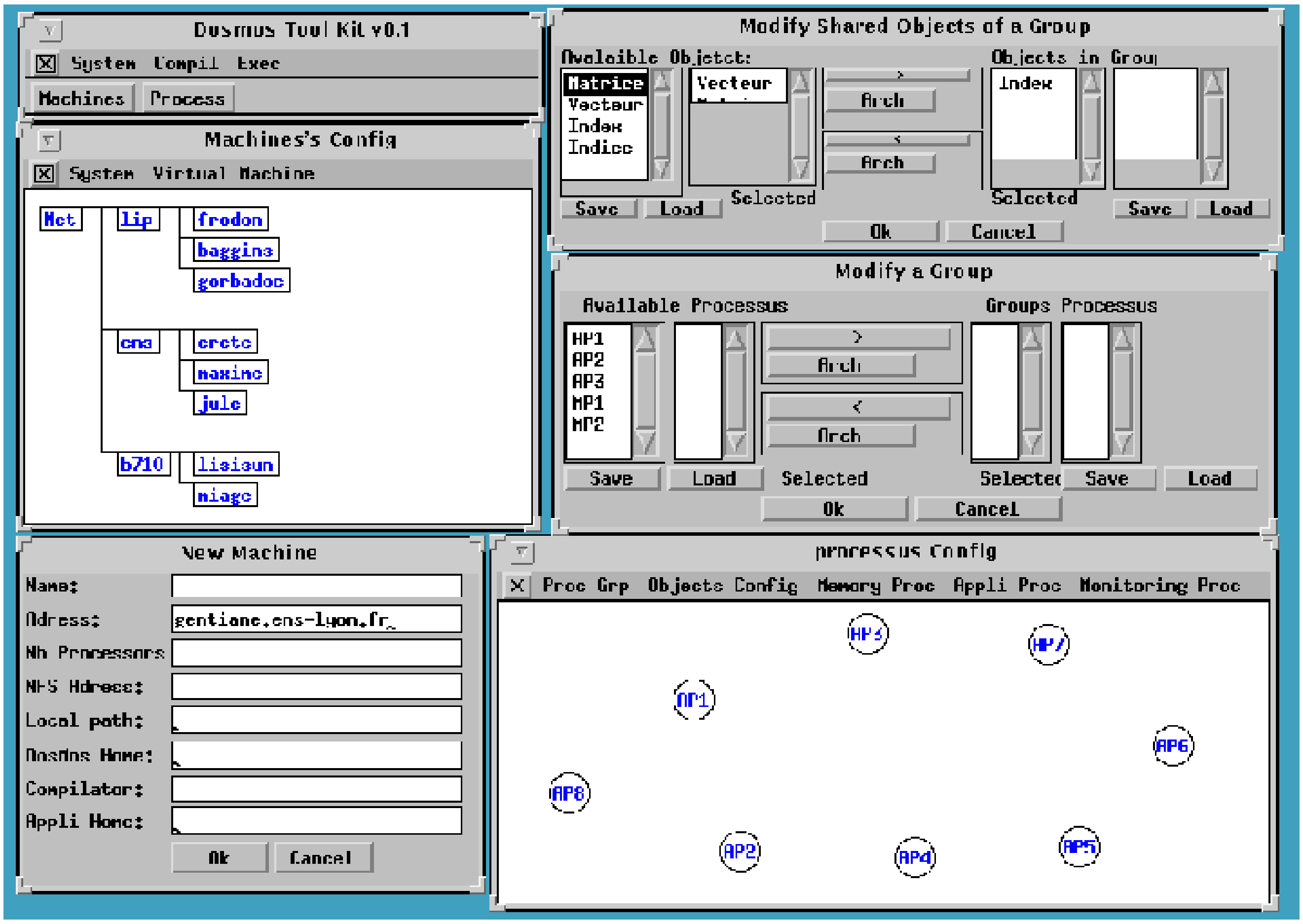

Figure 7. DOSMOS Tool Kit

[HERV93] Gernot Heiser, Kevin Elphinstone, Stephen Russell, and Jerry Vochteloo. Mungi: A distributed address-space operating system. Technical report, School of computer science and engineering. The university of New South Wales, November 1993.

[Lef96] Laurent Lefèvre. Parallel programming on top of DSM Systems : An Experimental Study. In To be published in SIAM, editor, *Environments and Tools for Parallel Scientific Computing III*, August 1996.

[Li88] Kai Li. IVY: A shared virtual memory system for parallel computing. In *1988 International Conference on Parallel Processing*, volume II, pages 94–101, August 1988.

[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.

[RAK89] Umakishore Ramachandran, Mustaque Ahamad, and M. Yousef A. Khalidi. Coherence of distributed shared memory: unifying synchronization and data transfer. In *1989 International conference on parallel processing*, volume II, pages 160–169, 1989.

[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.