A DSM-based Structural Programming Environment for Distributed and Parallel Processing

Lionel Brunie, Laurent Lefèvre

▶ To cite this version:

Lionel Brunie, Laurent Lefèvre. A DSM-based Structural Programming Environment for Distributed and Parallel Processing. HiPC '96 : 3rd International Conference on High Performance Computing, Dec 1996, Trivandrum, India, India. pp.469-474. hal-00767645

HAL Id: hal-00767645
https://inria.hal.science/hal-00767645v1

Submitted on 7 May 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A DSM-based structural programming environnment 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

Tel : 72 72 80 00 Fax : 72 72 80 80

(lbrunie, llefevre)@lip.ens-lyon.fr

Person who will present the work : Laurent LEFEVRE

Abstract

The paper describes an original programming environnment 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 schemes and protocols, we improve the performances of DSM system and provide an original programming model that mixes message-passing and shared distributed objects. Morevoer by integrating an easy development platform and montoring facilities, the DOSMOS system has been designed to provide a performant user-friendly programming environment.

1 Introduction

If distributed memory MIMD machines allow very high performances, their programming remains "esoteric" for most of end-users accustomed to classical mono-processor programming. Oppositely, shared memory parallel computers are easier to program but badly adapted to applications generating many memory accesses. In this context, the purpose of Distributed Shared Memory systems (DSM) is 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 the communications and to maintain the coherence of the shared data.

In that framework, this paper describes an original programming environment, called DOSMOS[1] system. This system is based on a structural approach of parallel programming. In other words, DOSMOS proposes to the user to hierarchically structure the processes into *groups* and *sub-groups of processes* sharing a same set of variables. This feature, combined with optimized 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.

However, it would be unrealistic to argue that DOSMOS, or any other DSM system, efficiently deals with any kind of applications. That is why DOSMOS allows the programmer to mix both message-passing (PVM) and DOSMOS code. To complete the programming environment, DOSMOS integrates a devoted monitoring tool (called DOSMOS-Trace) which has been added to the system to allow the user to understand the behavior of his 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 (as well as DOSMOS-Trace) has been developed on top of PVM.

[1] DOSMOS : Distributed Objects Shared MemOry System

This paper is divided into three parts. After a short description of previous works (section 2), we analyse the basics of the DOSMOS DSM system in terms of management of shared data and process structuring (section 3). Then a description of the programming environment is proposed (development platform, programming model and monitoring facilities). 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 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. Such systems like MIRAGE[FP89] 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 two types 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.

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.

Optimized 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) :

- *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 :

$$Global\ Virtual\ Space(P)\ =\ Local\ Virtual\ Space(P)\ +\ Extern\ Virtual\ Space(P)$$

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 $O \in LVS(P)$.

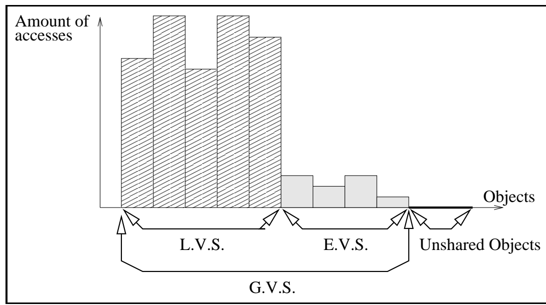

Figure 1: Example of accesses distribution

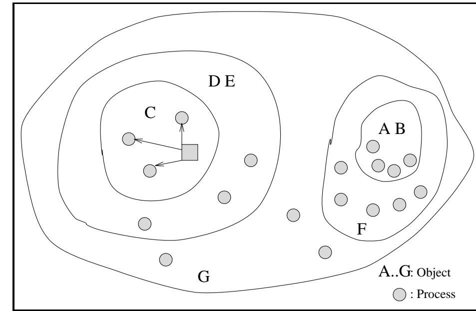

Figure 2: Hierarchical grouping of processes

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.

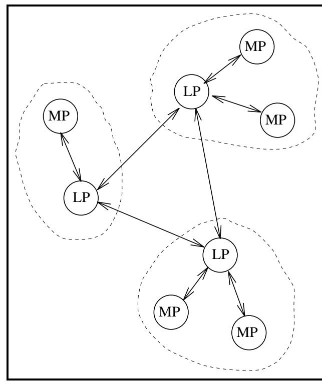

Figure 3: Groups and link processes

Experiments done with DOSMOS system on network of workstations and parallel machines have shown great improvements when using hierarchical groups.

4 Programming environment

4.1 Development platform

The implementation of DOSMOS is based on two different layers (see Figure 4):

- **Preprocessing level**: this layer analyses the user's application in order to detect and generate accesses to shared objects (fig. 5). This layer allows the system to be "transparent";
- **DSM level**: this layer assumes the creation and management of shared objects, groups and of various processes involved in execution of the application (APs, MPs, LPs). This management is performed using PVM routines.

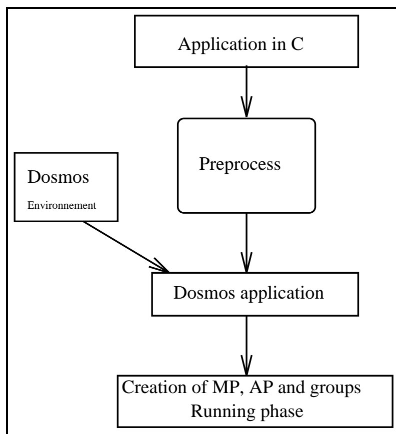

Figure 4: Dosmos Environment

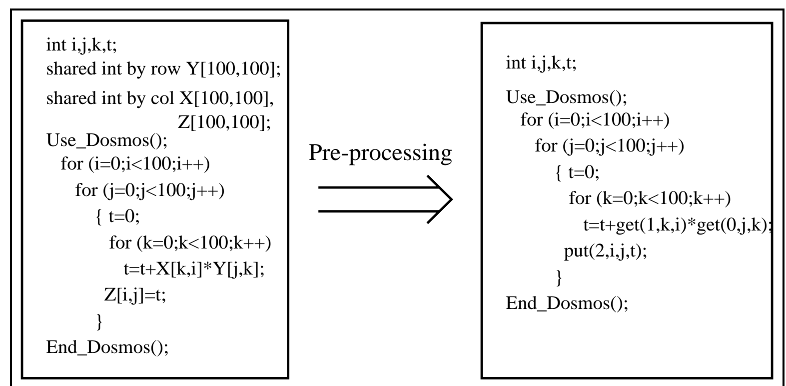

Figure 5: Example of pre-processing on a matrix multiplication application

4.2 DOSMOS primitives

By only adding a few new primitives, DOSMOS systems stays really easy to use for the user. 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*`)`, `acquire_blk(`*block*`)`
	`release(`*object*`)`, `release_blk(`*block*`)`
• Synchronization	`sync(`*object or group*`)`

Figure 6: DOSMOS primitives

4.3 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.

4.4 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.

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 (e.g. fig. 8), over-accessed variables, bottlenecks,

not actually shared variables, etc.

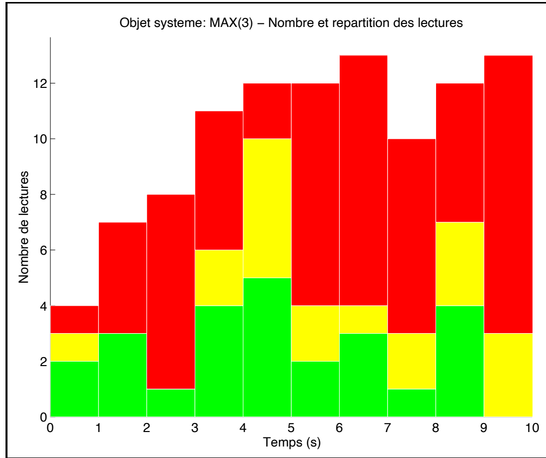

Figure 7: Number and origin of the read accesses performed on an object vs execution time (in black : inter-group accesses)

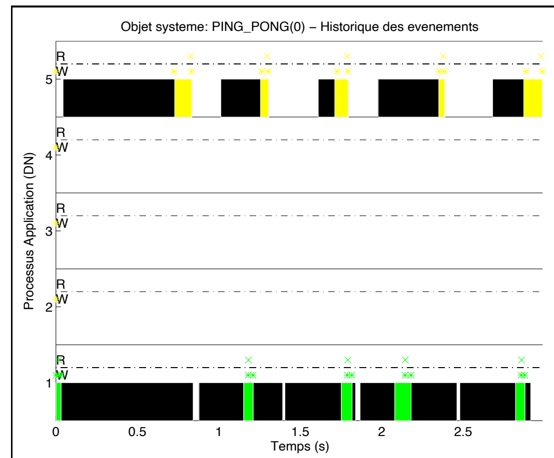

Figure 8: Object activity vs execution time : here a ping-pong effect between two processes.

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, possibility of mixing message-passing code and DSM code, optimized weak consistency protocols, monitoring facilities.

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. For the same purpose, the DOSMOS-Trace environment is based on distributed processes and has been designed to permit to use distributed traces files. Tests have shown the effectiveness of the approach developed in DOSMOS.

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 analyse execution application, DOMOS is an user-friendly programming environment which can easily adapt to a non-expert parallel programming user.

We currently continue to improve our programming environment by adding a new distributed tool to DOSMOS which will allow to debug the application code of the user. Moreover important DSM-applications (scientific computing, imagery, neural networks...) are currently being implemented on DOSMOS system to show the performance and effectiveness of our approach.

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. June 1996. to be published in the proceedings of the IEEE ICA3PP'96 conference (Singapore).

[BLR96] Lionel Brunie, Laurent Lefèvre, and Olivier Reymann. Execution analysis of DSM applications: A distributed and scalable approach. May 1996. to be published in the proceedings of the ACM SPDT'96 conference (Philadelphia, USA).

[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.

[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.

[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.