Dealing with Heterogeneity in Stardust: An Environment for Parallel Programming on Networks of Heterogeneous Workstations

Gilbert Cabillic and Isabelle Puaut

IRISA, Campus Universitaire de Beaulieu, 35042 Rennes Cédex, FRANCE
e-mail cabillic/puaut@irisa.fr

Abstract. This paper describes the management of heterogeneity in Stardust, an environment for parallel programming above networks of heterogeneous machines, which can include distributed memory multicomputers and networks of workstations. Applications using Stardust can communicate both through message passing and distributed shared memory. Stardust is currently implemented on an heterogeneous system including an Intel Paragon running Mach/OSF1 and an ATM network of Pentiums running Chorus/classiX.

1 Introduction

The proliferation of inexpensive and powerful workstations has continued to increase at a rapid rate in the last few years. This increase of machine performance is likely to continue for several years, with faster processors and multiprocessor machines. Studies have shown that for a large percentage of their lifetime, the machines are used for small tasks, thus demonstrating an average idle percentage of at least 90% even during peak hours. One possible use of these idle cycles is to run parallel applications. A number of research activities have tried to exploit the computing power of networks of workstations, like PVM [1] and MPI [2], based on the message-passing paradigm, and Mairmaid [3] based on the shared-memory (DSM) paradigm. To be fully usable, an environment for parallel computing above networks of workstations should: (i) support hardware and software heterogeneity, (ii) provide multiple programming paradigms, (iii) include mechanisms for load balancing and application reconfiguration, and (iv) tolerate machine failures.

Stardust is an environment that provides such facilities. Stardust allows the execution of parallel applications based both on the message-passing and page-based DSM paradigm. It executes on an heterogeneous computing environment composed of a parallel machine (the Intel Paragon) and an ATM network of workstations (PCs). Stardust includes a support for load balancing and checkpointing (see [4, 5] for more details). This paper focuses on Stardust support for heterogeneity. Section 2 gives an overview of the management of heterogeneity in Stardust. Section 3 then gives some performance data. Concluding remarks are given in section 4.

2 Dealing with Heterogeneity in Stardust

Stardust includes transparent mechanisms for converting data between different types of hosts. These mechanisms are used for converting both the contents of buffers exchanged in messages and the contents of shared virtual memory regions; they are described in paragraph 2.1. Furthermore, in order to cope with the processors different instruction formats, each program developed on top of Stardust is compiled for every type of architecture. Issues other than data conversion have to be dealt with, for processes to communicate through DSM (see [3, 4] for an overview of these issues). The main design decision for addressing these issues in Stardust consists in choosing a common unit for data conversion and data transfer between hosts, called *heterogeneous page* (see paragraph 2.2).

2.1 Mechanisms for data conversion and data typing

Stardust uses a standard architecture independent format for communications between different types of hosts. When sending a message, its contents is first converted into the standard data format; the receiver of the message then converts the message contents into its own physical format. The standard data format used for data transfer is the SUN eXternal Data Representation (XDR) format [6]; it was chosen because of its availability on a wide range of architectures and operating systems. The XDR library, linked with the Stardust environment, provides routines for converting basic data types to/from the XDR format.

In order to apply the XDR conversion routines when a piece of data is transferred between hosts of different types, it must be possible to have the type of each data structure. One approach consists in analyzing the source language (for instance by modifying the C compiler). Another approach, taken in [3] is to exploit information generated by compilers in object files, and intended to be used by symbolic debuggers (Symbol TABle, or stab). In Stardust, we did not want to modify the compiler or make assumptions on the kind of run-time information it generates in object files. Consequently, the type of every data structure is given explicitly by the application programmer when the data structure is allocated (region creation, allocation of a communication buffer). This is done by using a simple language, taken from [7] which is analyzed when the data structure is transferred between different types of hosts. A type is of the form *(TypeString*, N_1, N_2, ... , N_n*)*, where *TypeString* is the string identifying the data type (see syntax below), and N_i is the number of elements of the ith nested subtype of string *TypeString*.

```
Type      :  '{' (BasicType | Type)+ '}'
BasicType : 'C' | 'I' | 'L' | 'F' | 'D'
```

For example, the type *("{C{D}}",10,20)* corresponds to a data structure which is an array of 10 structures of type *t_str*, each structure being composed of a character and 20 double precision floating point values.

Let us call *base element* the first nested subtype of a data structure (in the above example, the base element is the type *t_str*). Note that due to the method used to associate a type to a data structure, the conversion algorithm can only apply to a multiple number of base elements.

2.2 Mechanisms for sharing virtual memory regions

Mainly two issues have to be addressed when building an heterogeneous DSM. Firstly, the different hosts can have distinct page sizes. Secondly, depending on the architecture, the number of base elements in a page can vary, because of the differences in the physical representation of data. Let us consider for example an application running on an Intel Paragon and on a Dec Alpha-based architecture. Assume that the application is sharing a vector whose type is *("{LC}",8192)*. Due to the architectures' characteristics and to the alignment constraints, the size of the base element on the Paragon is 6 bytes (4 for the long integer, 1 for the character and an extra-byte for the alignment constraints), while its size is 10 bytes on the DEC alpha, where long integers take 8 bytes. Even if the page sizes on the two architectures are identical (8 Kbytes), they do not contain the same number of base elements. On this example, they do not even contain an integral number of base elements. To our knowledge, no heterogeneous DSM fully addresses all the issues coming from heterogeneity. In [3], it is assumed that all basic data types have the same size on all architectures; in addition, the problem of a base element crossing a page boundary is solved by adding an additional alignment constraint so that this problem does not occur.

In Stardust, the issues of different data representations and page sizes are addressed by choosing a common unit for data conversion and data transfer between different types of hosts, called *heterogeneous page*. The size of an heterogeneous page is a multiple of the size of a base element, so that the same data item is never on the memory of the two hosts. In addition, the size of an heterogeneous page is a multiple of the page size of every architecture, so that an action on an heterogeneous page can always be mapped oñ a set of actions on pages of every architecture. In the above example, an heterogeneous page contains 4096 base elements, and corresponds to 5 Paragon pages and 3 Dec pages. Note that the decomposition of a shared region into heterogeneous pages not only depends on the page sizes of all the machines hosting the application. It is also dependent on the type of data structure contained in the region. If the region taken as example above had contained characters instead of a more complex data structure, the size of the heterogeneous page would have been different (a single virtual memory page for both architectures).

Fragmentation into heterogeneous pages: A region is fragmented into heterogeneous pages at region creation time. At this time, all the types of architectures on which the application is running are known, as well as the type of the data structure contained into the region. The fragmentation algorithm finds out, for every architecture, a page size for which no base element crosses a page boundary, and then takes the least common multiple (LCM) of theses values, thus obtaining the size of the heterogeneous page for the region.

Consistency protocol: Consistency of shared data is managed by a 2-level system. The *intra-architecture* level manages data consistency for a subset of homogeneous hosts. It uses a *homogeneous memory manager* (HoMM) per node. HoMMs manage data consistency within the associated architecture. They use the architecture virtual memory page as a unit of data transfer between machines. Pages are transferred using the most efficient communication protocol that exists on the architecture. In addition, pages are transferred in the architecture physical data representation, without conversion into an architecture-independent format.

In the *inter-architecture* level, data consistency is managed by an *heterogeneous memory manager* (HeMM) per group of nodes of the same type. The unit of data transfer between HeMMs for a given region is the heterogeneous page that corresponds to the region's type. Transfers are achieved via a common communication protocol (TCP) and data is transferred in the XDR architecture independent format. In the current implementation, both HoMMs and HeMMs implement sequential consistency through a write-invalidate consistency protocol, and use K Li's static distributed scheme for locating shared pages [8].

Note that with such a structure of heterogeneous DSM there is no time overhead due to the management of heterogeneity when an application is running on hosts of the same type.

3 Implementation and Performance of Stardust

Stardust is currently implemented on a 56 nodes Intel Paragon [9] running a Mach/OSF1 kernel and on an 155 ATM network of Pentium PC machines[1] running the Chorus/classiX operation system [10]. Each Paragon node is equipped with 16 Mbytes of memory, of which nearly 8 Mbytes are consumed by the operating system. The size of pages on the paragon is 8 Kbytes. The measured transfer rate between nodes is 60 Mbytes/s. Each PC has 32 Mbytes of memory, of which only 8 Mbytes are left free for the paging activity. The size of pages on the Pentiums is 4 Kbytes. For all the performance measures given in this paragraph, the size of heterogeneous pages is 8 Kbytes.

Stardust is made up of a set of software modules, which have an OS independant interface in order to ease their portability. The *consistency manager* maintains data consistency using a write-invalidate protocol ; the same code is used by HoMMs and HeMMs. The *network manager* offers primitives for exchanging messages. It uses for intra-architecture communications are running the most efficient communication protocol of the architecture (NX communication library on the Intel Paragon, Chorus IPC on the Pentium PCs). If the message sender and receiver run on different types of hosts, TCP is used via sockets, and the message is converted into the XDR format before being sent. Finally, the *OS*

[1] The ATM driver being currently under debug, this version of the paper includes figures measured on a 10Mb/s Ethernet.

interaction manager is responsible of the communication with the underlying operating system concerning memory management and thread management.

Figure 1 shows the performance of the *MGS* (Modified Gram-Schmidt) application, which produces from a set of vectors an orthonormal basis of the space generated by these vectors. The problem size for this application is 256x1024 double precision floats. The application only uses DSM and does not exhibit false sharing. The figure shows the elapsed time for the application when it runs on (i) a set of Paragon nodes, (ii) a set of Pentium nodes, and (iii) an heterogeneous system with the same number of nodes of both architectures. Measurements were done in a system with 1 up to 8 nodes.

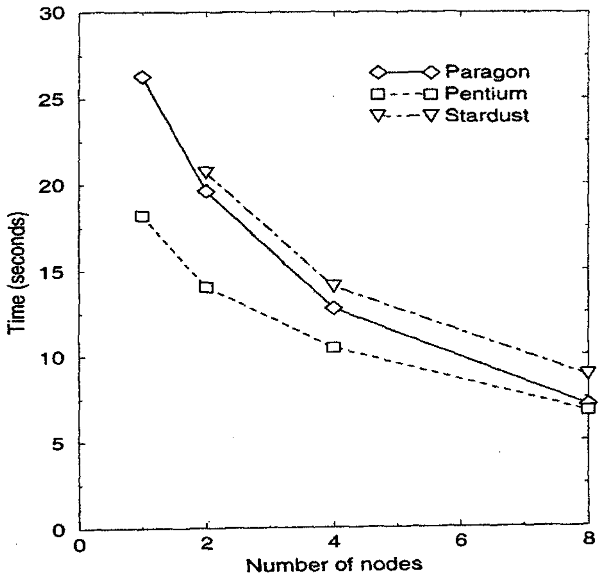

Fig. 1. Execution time of MGS application

The curves show that the application launched on an homogeneous set of Pentium nodes always exhibits better performance compared to the same application running on Paragon nodes or on heterogeneous nodes. In addition, the applications still exhibits a good speedup when the number of nodes increases. Finally, only a small percentage of the total execution time is needed to deal with heterogeneity compared to the execution time for the Paragon machines.

4 Concluding Remarks

Many environments for parallel programming on networks of heterogeneous workstations have been designed and implemented in the last ten years. A key difference between Stardust and most of these environments is that Stardust runs both on parallel machines and networks of workstations, and includes mechanisms for load balancing and checkpointing [4, 5]. In addition, unlike most environments, Stardust supports both the shared-memory and message-passing paradigm. From an implementation point of view, we have tried in Stardust to use the most efficient communication protocol of a given architecture for communications between hosts of this architecture. In comparison, most implementations of PVM use TCP for all communications, which can be less efficient than architecture-specific communication protocols. Few DSM systems have been designed for heterogeneous systems. Marmaid [3] differs from Stardust by the way types are associated to shared memory regions. Mairmaid requires less effort from the programmer, but its implementation depends on the format of the object files. Unlike Mairmaid, it is not assumed in the Stardust prototype that basic data types have the same size on all architectures.

References

1. A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam. *PVM: Parallel Virtual Machine - A Users' Guide and Tutorial for Networked Parallel Computing.* MIT Press, 1994.
2. W. Grop, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message-Passing Interface.* MIT Press, 1994.
3. S. Zhou, M. Stumm, K. Li, and D. Wortman. Heterogeneous distributed shared memory. *IEEE Transactions on Parallel and Distributed Systems*, 3(5):540–554, September 1992.
4. G. Cabillic and I. Puaut. Stardust: an environment for parallel programming on networks of heterogeneous workstations. Technical Report 1006, IRISA, April 1996. Available by anonymous ftp at ftp.irisa.fr.
5. G. Cabillic, G. Muller, and I. Puaut. The performance of consistent checkpointing in distributed shared memory systems. In *Proc. of the 14th Symposium on Reliable Distributed Systems*, pages 96–105, Bad Neuenahr, Germany, September 1995.
6. Sun Micrósystems Inc. *Network Programming Guide - External Data Representation Standard: Protocol Specification*, 1990.
7. C. Pinkerton. A heterogeneous distributed file system. In *Proc. of 10th International Conference on Distributed Computing Systems*, May 1990.
8. K. Li and P. Hudak. Memory coherence in shared virtual memory systems. *ACM Transactions on Computer Systems*, 7(4):321–357, November 1989.
9. Intel. *Paragon User's Guide*, 1993.
10. M. Rozier, V. Abrossimov, F. Armand, I. Boule, M. Gien, M. Guillemont, F. Herrmann, P. Léonard, S. Langlois, and W. Neuhauser. The Chorus distributed operating system. *Computing Systems*, 1(4):305–370, 1988.