19. Programming SCI Clusters Using Parallel CORBA Objects

Thierry Priol[1], Christophe René[1], Guillaume Alléon[2]

[1] IRISA/INRIA, Campus de Beaulieu, 35042 Rennes Cedex, France
email: {priol, crene}@irisa.fr
http://www.irisa.fr/
[2] Aerospatiale Joint Research Centre, 12 rue Pasteur, BP 76,
92152 Suresnes Cedex, France
email:Guillaume.Alleon@siege.aerospatiale.fr

19.1 Introduction

This chapter introduces a programming environment for SCI clusters that takes advantage of both parallel and distributed programming paradigms. It aims at helping programmers to design high performance applications based on the assembling of generic software components. This environment is based on CORBA (Common Object Request Broker Architecture), with our own extensions to support parallelism across several cluster nodes within a distributed system. Our contribution concerns extensions to support a new kind of object, which we call a parallel CORBA object (or parallel object), as well as the integration of message-passing paradigms, mainly MPI, within a parallel object. These extensions exploit as much as possible the functionality offered by CORBA and require few modifications to an available CORBA implementation. This paper reports on these extensions and the description of a runtime system, called *Cobra*, which provides resource allocation services for the execution of parallel objects.

The chapter is organized as follows. Section 19.2 discusses some issues related to parallel and distributed programming. Section 19.3 gives a short introduction to CORBA. Section 19.4 describes the concept of parallel CORBA objects. Section 19.5 introduces the *Cobra* runtime system for the execution of parallel objects. Section 19.6 presents a case study based on a signal processing application from Aerospatiale. Section 19.7 describes some related work which has some similarities with our work. Finally, Section 19.8 draws some conclusions and outlines perspectives of this work.

19.2 Parallel vs. Distributed Programming

Thanks to the rapid performance increase of today's computers, it can be now envisaged to couple several computationally intensive numerical codes to simulate more accurately complex physical phenomena. Due to both the increased complexity of these numerical codes and their future developments, a

H. Hellwagner, A. Reinefeld (Eds.): SCI'99, LNCS 1734, pp. 333-348, 1999

tight coupling of these codes cannot be envisaged. A loose coupling approach based on the use of several components offers a much more attractive solution. One can envisage to couple fluid and structure components or thermal and structure components. Other components can be devoted to pre-processing (data format conversion) or post-processing of data (visualization). Each of these components requires specific resources (computing power, graphics, specific I/O devices). A component which requires a huge amount of computing power can be parallelized so that it will be seen as a collection of processes to be run on a set of cluster nodes. Processes within a component have to exchange data and have to synchronize. Therefore, communication has to be performed at different levels: between components and within a component. However, the requirements for communication between or within a component are quite different. Within a component, since performance is critical, low-level message-passing is required, whereas between components, although performance is still required, modularity/interoperability and re-usability are necessary to develop cost effective applications using generic components.

However, until now, most programmers who are faced with the design of high-performance applications use low-level message-passing libraries such as MPI or PVM. Such libraries can be used for both coupling components and for handling communication among processes of a parallel component. It is obvious to say that this approach does not contribute to the design of applications using independent software components. Such communication libraries were developed for parallel programming; they do not offer the necessary support for designing components which can be reused by other applications.

Solutions already exist to decrease the design complexity of such applications. Distributed object-oriented technology is one of them. A complex application can be seen as a collection of objects which represent the components, running on different machines and interacting using remote object invocations. Emerging standards, such as CORBA, support the design of applications using independent software components through the use of CORBA objects. For the rest of the chapter, we will use the term object to name a CORBA object. CORBA is a distributed software platform which supports distributed object computing. However, exploitation of parallelism within such an object is restricted in a sense that it is limited to a single node within a cluster. CORBA implementations such as Orbix from Iona Technologies [8], allow the design of multi-threaded objects that can exploit several processors within a single SMP (Symmetric Multi-Processing) node. Such an SMP node cannot offer the large number of processors which is required for handling scientific applications in a reasonable time frame. However, the required number of processors is available at the cluster level where several dozens of machines are connected. Nevertheless, application designers have to deal "manually" with a large number of objects that have to be mapped onto different nodes of a cluster, and to distribute computations and data among these objects.