Developing Applications for Multicomputer Systems on Workstation Clusters

Georg Stellner, Arndt Bode,
Stefan Lamberts and Thomas Ludwig*

Technische Universität München
Institut für Informatik
Lehrstuhl für Rechnertechnik und Rechnerorganisation
D-80290 München
nxlib@informatik.tu-muenchen.de

Abstract. Much computational power on state-of-the-art multicomputers like the Paragon is wasted with porting applications. Using networks of workstations is an attempt to withdraw this workload from multicomputer systems. Therefore an environment is needed which provides the programming interface of multicomputers on coupled workstations. The paper describes the design and implementation of the NXLib environment which allows to use Ethernet coupled workstations as a development platform for applications targeted for Intel Paragon systems.

1 Motivation

A drawback of multicomputers is that porting existing applications onto those systems often requires enormous efforts. Applications have to be parallelized which leads to frequent test runs during the implementation. Therefore, much workload on multicomputer systems consists of test and debugging runs. To withdraw some of this load, an environment is needed which allows the implementation of applications for multicomputer systems on different hardware platforms.

Today, typical environments in universities and companies consist of several networked workstations. The basic architecture of multicomputer systems and coupled workstations is similar: independent processing elements (nodes or workstations) which are interconnected. In difference to the multicomputers' high performance interconnection network, workstations currently use a slower interconnect. In addition the network has to be shared with other machines and users which are also connected to the network.

State-of-the-art multicomputers like the Paragon offer a proprietary message passing environment. An implementation of that library on coupled workstations would allow for using interconnected workstations as a development platform for applications where the production code should finally run on a multicomputer

* This project was partially funded by a research grant from the Intel Foundation.

system. In addition to that, it is also applicable to use interconnected workstations as additional computational resource. The performance constraints of coupled workstations restricts this to applications with limited demands concerning computational power and a coarse grained or medium grained granularity of parallelism.

In the following we will describe the design and implementation of the Paragon NX communication library for workstation nets. Therefore we first give a short description of the Paragon and its software environment. After that, we introduce the design and implementation of the NXLib package for coupled workstations. Some performance figures of the current NXLib release are provided in chapter 4. Finally, the last chapter summarizes and gives an outlook on future work.

2 The Paragon and its Message Passing Interface

To get a better understanding of NXLib's design and implementation, we first present a short overview on the Paragon and the NX message passing library [1]. The nodes of a Paragon system are interconnected in a two-dimensional mesh topology, which is subdivided into three partitions: the I/O partition, the service partition and the compute partition (see fig. 1). Usually the largest partition in a

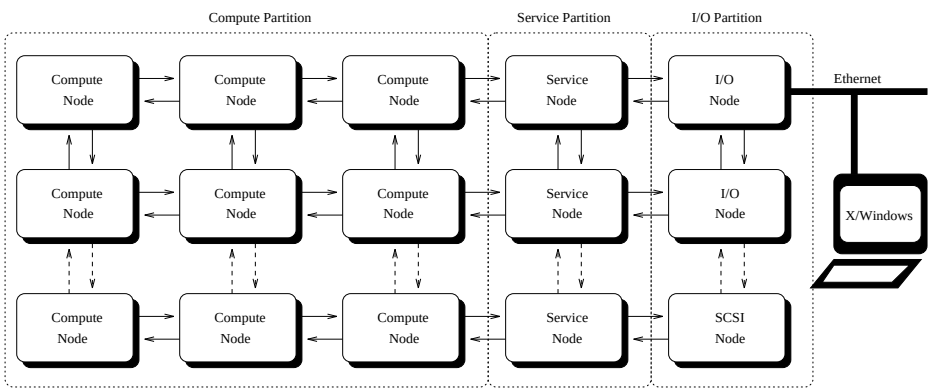

Fig. 1. Different partitions in a Paragon system

configuration is the compute partition. Parallel user applications are executed on the nodes in this partition. In contrast to that, interactive processes are executed on the nodes in the service partition. Finally, the nodes in the I/O partition are used to connect I/O devices.

Parallel user applications on the compute partition make use of Intel's message passing library which is derived form the NX/2 of the iPSC systems [3]. Apart from synchronous, asynchronous and interrupt-driven communication calls, NX provides calls for process management.

3 Design and Implementation of NXLib

In the following sections a short introduction to the design and implementation of NXLib is given. For a more detailed discussion refer to [5].

3.1 The node model

In the following the meaning of some frequently used terms will be explained. A parallel application on a Paragon system consists of two parts. The application processes on the compute partition and the controlling process of the application on one node of the service partition. In the following discussion the term *Paragon node* will be referred to as the collection of a hardware Paragon node, the operating system kernel and a set of application processes running on top of that.

The basic means to model Paragon nodes on coupled workstations is virtualization. Consequently, the term *virtual Paragon node* (VPN) describes a *Paragon node* on a workstation. The hard- and software properties of a *Paragon node* which are not available on a workstation are virtualized in the following way. A natural approach to model them is to introduce a daemon process which virtualizes the node hardware and the operating system. The calls of the application processes to NX communication routines are transformed into requests to the the daemon. In such an implementation every system call would require an

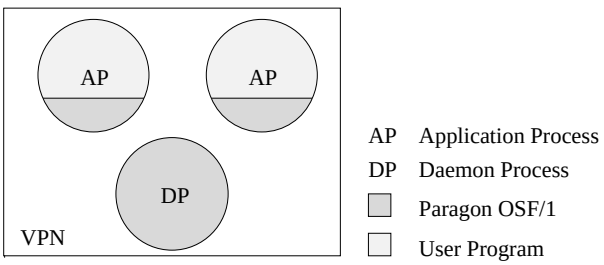

Fig. 2. Processes and the distribution of the operating system on a VPN

interprocess communication. To reduce the amount of interprocess communication parts of the operating system's tasks have been moved into the application processes like illustrated in fig. 2.

3.2 Layers of NXLib

An important issue for a message passing library for coupled workstations is portability and flexibility. A layering of the message passing library has been designed to cover both aspects. Figure 3 shows the layers of the NXLib environment. The basis form the standard UNIX system calls. To achieve a great flexibil-

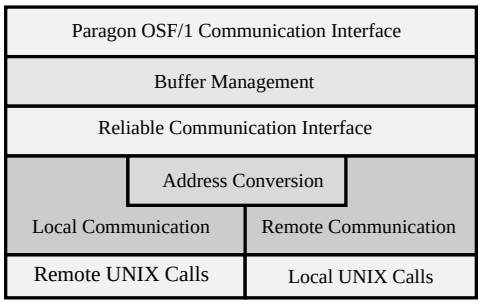

Fig. 3. Layers of the NXLib environment

ity concerning the communication protocol which is used for the implementation NXLib distinguishes between local and remote communication. Within the local and remote communication layer a protocol specific addressing scheme is used. The reliable communication layer provides reliable point-to-point communication calls disregarding the location of the communication partners. The reliable communication interface still uses the Paragon addressing scheme. The address conversion layer has been introduced to map Paragon addresses to corresponding protocol specific addresses. In addition to its address conversion task this layer also distinguishes whether a communication is local or remote. Provided with that information the reliable communication layer can invoke the appropriate local or remote communication calls.

The Paragon OSF/1 communication interface finally provides the user calls which are available on a Paragon system. The calls of the buffer management to insert and delete messages into the message table are used to map messages to corresponding user calls. All user communication calls interface the communication system via the message table.

3.3 Modeling Paragon partitions

In addition to the partitions which were introduced in section 2 it is also possible to define sub-partitions of the compute partition. In a workstation environment mapping files can be used to simulate such partitions. Within that file a mapping of virtual node numbers to workstations is provided. Thus, the mapping table defines a *virtual compute partition*.

A problem occurs for the service partition. It is not part of the Paragon partition management which is available for the user. Consequently a different means has to be provided to establish a *virtual service partition*. This is simply done by defining the machine where the application has been started as the *virtual service partition* of the *virtual Paragon* on the workstations.

3.4 NX message passing calls on workstations

An important issue for message passing libraries is the performance of the communication calls. Both local and remote communication use TCP sockets because

this protocol achieves high throughput rates. To reduce the latency it is desirable to use direct paths between communication partners. Every stage in an indirect scheme increases the latency as additional calls have to be performed. On the other hand, on most UNIX systems the number of socket descriptors is limited. A full interconnection of all application processes would therefore drastically reduce the number of processes in an application. Establishing and terminating a communication link between two processes for every communication call is not feasible either as this would introduce much additional effort for every communication.

The basic assumption of our implementation is that typical parallel applications have a regular communication structure in the sense that certain processes regularly communicate with each other. Thus, two processes are either connected and use this communication path frequently during the computation or they do not communicate at all. Consequently, communication paths need only to be created for those processes that wish to communicate. As the communication structure of an application can not be determined at start time, the interconnection of the processes can certainly not be done during the initialization of the application. So the communication paths between processes are set up on demand during run time. Once established a connection between two processes is kept until the application terminates. Building up the connections on demand has the advantage that all interacting processes are fully interconnected. So communication latencies can be kept minimal for established communication links. Finally, as only those processes are interconnected which need to communicate more processes can participate in an application. The only drawback is that the first communication between two processes is more expensive than the following because the connection has to be set up.

4 Applications and Performance

To evaluate the NXLib environment we have used two coarse grain applications which we have running on Paragon systems: NSFLEX [2] and MUMUS [4]. In both cases only minor changes to the makefiles were necessary to compile and link the source code. After the compilation the applications can be started like on a Paragon system by specifying the name of the executable at a shell prompt. To select a virtual partition the same command line switch like on a Paragon can be used. Instead of the partition name the mapping file has to be specified. In a similar way the number of processes which should be created during the start up can be specified with the appropriate Paragon command line switch.

The performance comparison is based on the solution of the same problems on both platforms. To achieve comparable results the problems were solved on a four node Paragon partition and on four Sun Sparc 10. These were the most powerful machines and the maximum number which were at our disposal. Computations on more machines, which included some Sun SLC, made obvious that the performance is driven by the slowest machine in the configuration. On the Paragon its OSF/1 release 1.0.3 was running whereas the Suns executed the SunOS 4.1.1

operating system kernel. With these operating systems a single Paragon node can achieve a floating point performance which is up to three times better than a single Sparc 10. Fig. 4 illustrates the results of the computations.

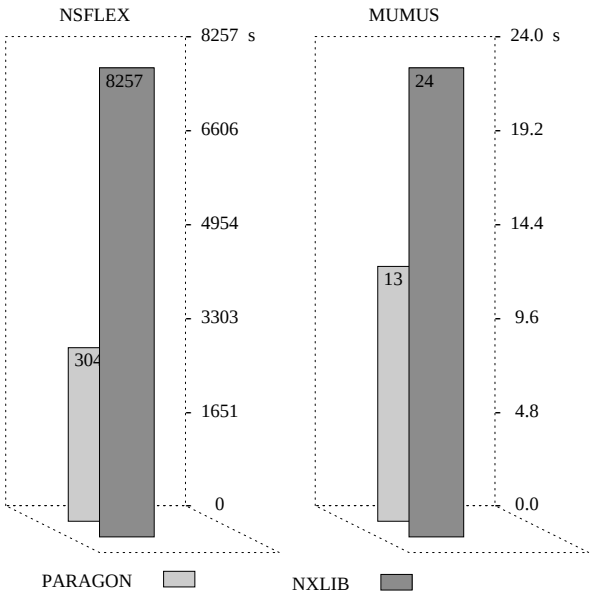

Fig. 4. Comparison of the execution times of MUMUS and NSFLEX on a Paragon and a network of workstations

For NSFLEX the Paragon system is nearly three times as fast as the workstations. For the computation of the given problem with MUMUS the workstations need about twice the time as the Paragon. Taken into consideration the performance of applications on coupled workstations using NXLib seems very promising. These results have to verified on larger clusters and more powerful machines than the Sparc 10.

5 Conclusion and Future Work

The NXLib environment allows for using a network of workstations for mainly two purposes. First, the network of workstations can be used to develop software which should finally run on a Paragon system. Workload can therefore be withdrawn from the multicomputer system. The CPU time which is gained by shifting the development of applications to workstations can be used for production runs of computational intensive problems. Second, instead of using the workstations merely as a development platform they can also be used as a production environment for certain applications. Especially coarse grain applications can achieve good speed-ups on a workstation environment.

Basically NXLib offers the same programming environment as a Paragon system. Virtualization is the basic means to achieve this. Therefore, source code which has been implemented using NXLib can be ported to a Paragon without any changes.

An important issue for scientific and commercial applications is the support of parallel I/O. Due to the restricted network bandwidth of bus coupled workstations it is not feasible to use a single disk as I/O facility. A more interesting approach would be to use the local disks of the workstations and to set up a virtual Paragon file system on these disks. Concepts for disk and file striping in such an environment must be examined therefore.

Up to now there is no support for the programmer during the implementation process of an application. Efficient coding is an important issue for software projects. Thus, the support of a tool environment which assists the programmer during all steps in the software life cycle is very desirable. Tools which can be used to visualize or debug parallel applications require the possibility to gather run-time information. This can either be done on-line with a monitoring system or off-line through trace files. In both cases an instrumentation of NXLib is necessary to produce the data.

References

1. Intel Supercomputer System's Division, 15201 N.W. Greenbrier Parkway, Beaverton, OR 97006. *Paragon OSF/1 C System Calls Reference Manual*, 1 edition, April 1993.
2. T. Michl, S. Maier, S. Wagner, M. Lenke, and A. Bode. Dataparallel Navier-Stokes Solutions on Different Multiprocessors. In ASE'93, editor, *Applications of Supercomputers in Engineering*, September 1993.
3. Paul Pierce. The NX/2 Operating System. In *Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications*, pages 384–391. ACM, 1988.
4. M. Schumann, M. Kiehl, and R. Mehlhorn. Performance Evaluation of NXLib Using Parallel Multiple Shooting. [6], pages 58–64.
5. G. Stellner, S. Lamberts, A. Bode, and T. Ludwig. Design and Implementation of NXLib. [6], pages 6–17.
6. G. Stellner, M. Schumann, S. Lamberts, T. Ludwig, A. Bode, M. Kiehl, and R. Mehlhorn. Developing Multicomputer Applications on Networks of Workstations Using NXLib. SFB-Bericht 342/17/93 A, Technische Universität München, 80290 München, December 1993.

This article was processed using the LaTeX macro package with LLNCS style