Paragon Parallel Programming Environment on Sun Workstations

Stefan Lamberts, Georg Stellner,
Arndt Bode and Thomas Ludwig*
Institut für Informatik
Lehrstuhl für Rechnertechnik und Rechnerorganisation
Technische Universität München
D-80290 München
{lamberts,stellner,bode,ludwig}@informatik.tu-muenchen.de

October 8, 1993

Abstract

Today's requirements for computational power are still not satisfied. Supercomputers on the one hand achieve good performance figures for a great variety of applications but are expensive to buy and maintain. Multiprocessors like the Paragon XP/S are cheaper but require more effort to port applications. As one consequence, much computational power of such systems is wasted with debugging these codes. An attempt to withdraw implementation and debugging codes from multiprocessor systems is the usage of coupled workstations. A software environment for networks of workstations allows for implementation and testing of applications. After having been tested the applications can then be shifted to the multiprocessor systems by recompilation. The paper describes the design and implementation of an environment which allows to use Ethernet coupled Sun SPARC systems as a development platform for applications targeted for Intel Paragon XP/S systems.

1 Motivation

Scientific and commercial applications require much computational power. Today's supercomputer systems have been developed to satisfy these demands. Their computational power is sufficient to solve some of the so called *Grand Challenge Problems*. Typically, these machines are very expensive and difficult to maintain, e.g. they need a water-cooling system. A different architectural approach has been made to reduce the costs for such powerful machines. Assembling cheap and simple standard components such as processors and memory chips into a single machine saves purchase as well as maintenance costs. These machines are the classical distributed memory multiprocessor systems where standard microprocessor nodes are interconnected with a high performance interconnection network. Intel's Paragon XP/S system is a typical member of this class.

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

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

Today, typical environments in universities and companies consist of several workstations all interconnected via standard Ethernet. The basic architecture of multiprocessor systems and coupled workstations is similar: independent processing elements (nodes or workstations) which are interconnected. In difference to the multiprocessors' high performance interconnection network, workstations 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 multiprocessors like the Paragon currently 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 multiprocessor system.

Message-passing libraries for coupled workstations which offer a user interface similar to a multiprocessor can withdraw workload from these systems. In addition to that, it is also applicable to use interconnected workstations as additional computational resource. During times of low system load on the workstations their aggregated computational power can be used to run production versions of applications.

Two restrictions apply for this approach. First, the computational power offered by a number of workstations in a local area network does not reach today's multiprocessor systems. And second, the communication speed of the interconnection network is several orders of magnitudes lower than the one of multiprocessor systems like the Paragon. Thus, suitable applications are restricted to those with limited demands concerning computational power and the granularity of parallelism should be medium or even better coarse.

In the following we will describe the design and implementation of the Paragon OSF/1 communication library for Sun workstations which are interconnected via Ethernet. Therefore we first give a short description of the Paragon, its operating system and the message-passing library NX. After that, we introduce the design of the NXLIB message-passing library for coupled workstations. In chapter 4 we show in detail how this concept has been implemented. The last two chapters finally give a summary and an outlook on future work.

2 The Paragon and its Message-Passing Interface

To get a better understanding of the design and implementation we have chosen, we first present a short overview on the Paragon, its OSF/1 operating system and the NX message-passing library. Intel's Paragon is a MIMD [1] system with distributed memory. Figure 1 shows the basic architecture of the Paragon nodes. Each node consists of two

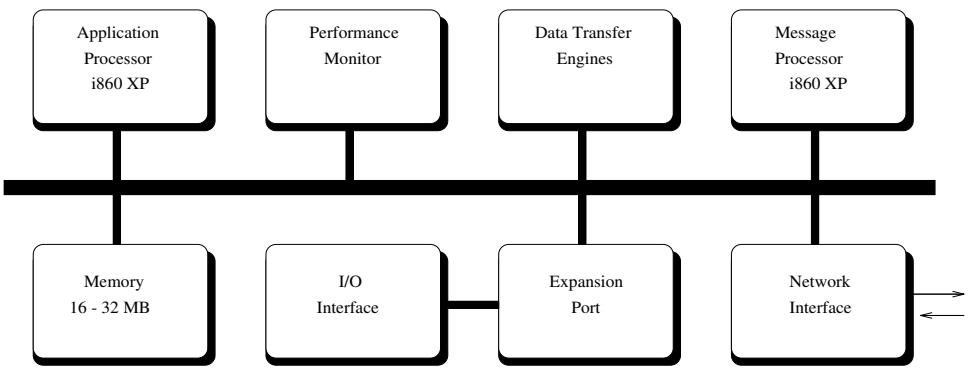

Figure 1: Architecture of the Paragon nodes

Intel i860/XP [6] microprocessors: one to run the operating system and user applications (application processor) and one to handle the communication between the nodes (message processor). Both processors access the local memory which can be up to 32MB large via a

common bus. A DMA controller (data transfer engine) allows for efficient data movement on each node. An additional expansion port and an I/O interface can be used to attach peripherals to a node. Finally a hardware monitoring chip has been integrated to provide low intrusion performance measurements on each node.

The nodes of a Paragon system are interconnected in a two-dimensional mesh topology. Each node is connected to a special routing chip, the so called iMRC. The iMRC chip routes

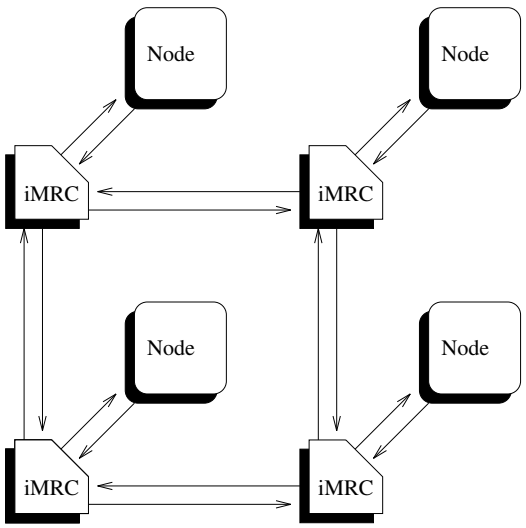

Figure 2: Interconnection scheme of a Paragon system

the messages between the nodes using a wormhole routing algorithm [7]. The links between the iMRC chips are 16 bit wide and achieve a bidirectional communication bandwidth of 350MB/s.

The nodes in a Paragon system are subdivided into three partitions: the I/O partition, the service partition and the compute partition. Figure 3 shows a typical configuration of a Paragon system. Usually the largest partition in a configuration is the compute partition.

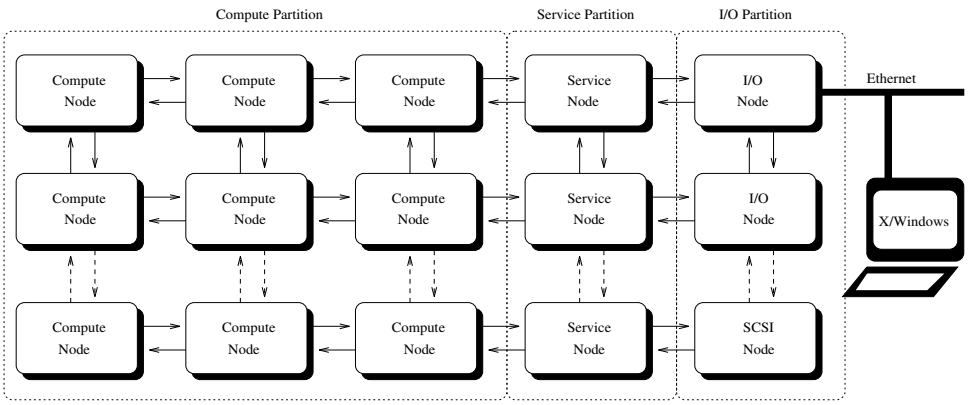

Figure 3: Different partitions in a Paragon system

Parallel user applications are executed on the nodes in this partition. In contrast to that, interactive processes, like shells, editors etc., are executed on the nodes in the service partition. Finally, the nodes in the I/O partition are used to connect I/O devices, like disks or local area networks, to the machine.

Although the nodes are arranged in different partitions they execute the same operating system kernel. The Paragon operating system is a Mach 3.0 based implementation of the

OSF/1 operating system [8]. It provides the user with a single system image of the machine. Any command which a user invokes during an interactive machine session is executed on any of the nodes in the service partition. Files can be transparently accessed from any node. File accesses are therefore transformed into corresponding requests to the nodes in the I/O partition.

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 [9]. Apart from synchronous, asynchronous and interrupt-driven communication calls, NX provides calls for the process management of parallel applications. Cooperating processes address each other via a node number and a process type (*ptype*). The node number is derived from the node where the process is executing, whereas the *ptype* can be modified via corresponding calls [3, 4, 2, 5].

The following section will introduce the concepts which were necessary to offer a similar system image on a network of workstations to the one available on the Paragon. This will include a more detailed discussion of some Paragon features where it seems appropriate.

3 The Design of the Paragon Message-Passing Library for Workstations

Due to the predefined user interface of the environment the design process of the NXLIB was limited to finding a model for the Paragon node, a layering of the software and a mapping of Paragon partitions to the workstations. Each of the following three sections in turn will give a short introduction to one of those topics.

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[1] and the controlling process of the application on one node of the service partition. Parallel applications require to be linked with a special linking option (–nx or –lnx), which includes the NX calls. Apart from the NX calls the application processes also can make use of the OSF/1 system calls. In the following discussion the term *Paragon node* will be referred to as the collection of a hardware Paragon node, the OSF/1 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 NXLIB software environment. The VPN is the smallest unit of distribution in the NXLIB environment, i.e. upon startup the user can define how many VPNs he wants to use and on which machine a specific VPN should be located. Aspects concerning the mapping of the VPNs to machines will be discussed in section 3.3. Currently a standard lightweight process library is not yet available on every UNIX system. Thus, the decision was made to use heavyweight UNIX processes to model a VPN on a workstation. Section 4.1 will show which processes are required to model VPNs and give a detailed description of their cooperation.

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 4 shows the layers of the NXLIB environment. The basis forms the standard

[1] As the Paragon allows the definition of hierarchical partitions the application processes may also execute on a sub-partition of the compute partition (see section 3.3).

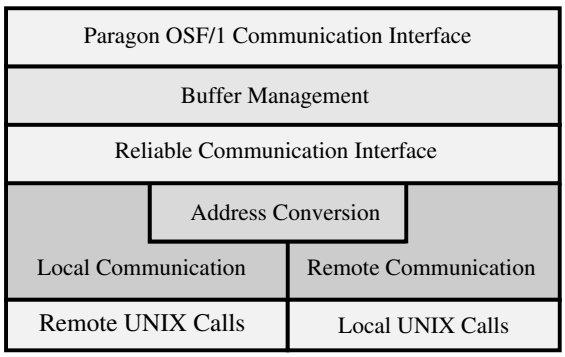

Figure 4: Layers of the NXLIB environment

UNIX system call layer with its different interprocess communication calls. To achieve a great flexibility concerning the communication protocol which is used for the implementation NXLIB distinguishes between local and remote communication. Thus, for either case it is possible to use a protocol which achieves the best performance. 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 consisting of a node number and a *ptype* 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 communication calls of either the local or remote communication layer.

On a Paragon system the OSF/1 operating system provides a sophisticated buffer management. Its parameters can be configured upon the startup of an application with several command line switches. This mechanism allows for adapting the usage of the limited memory resources on each node to the needs of an application in the best way. In addition, reserving enough buffer space may be required for certain applications to avoid deadlocks. A communication flow protocol has been included in the Paragon communication to avoid flooding a node's buffers with messages.

The buffer management layer we have introduced is based on the simplifying assumption that on each machine unlimited buffer space is available. Unlike the Paragon, where incoming messages are placed in a prereserved memory area, in NXLIB the memory is dynamically allocated when a message arrives on a node. Consequently a control flow protocol and the configuration parameters for the buffer sizes can be omitted in NXLIB.

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 calls are therefore not directly based on a communication but make use of calls which update the message table.

3.3 Modeling Paragon partitions

A short overview concerning the three basic partitions on a Paragon system was already provided in section 2. As an enhancement to the hardware definition of partitions the users can also define software partitions. These software partitions are compounded of any selection of nodes in the compute partition. The Paragon OSF/1 operating system provides calls to define and modify such partitions. Similar to the UNIX file system partitions have an owner (creator), access permissions, a name and may be created hierarchically.

In a workstation environment the situation is different. One way to provide a similar

semantics is to use mapping files. Within the file a table has to be specified to map virtual node numbers to workstation names or Internet addresses respectively. The owner, access permissions and name of the mapping table can be used to simulate the corresponding Paragon partition properties. In addition to that the file system hierarchy can be used to model the hierarchical definition of the partitions. 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.

4 The Implementation of NXLIB

In the previous section we have shown which concepts were developed to virtualize a Paragon system on a network of workstations. The next sections will show how these concepts were realized.

4.1 Implementation of virtual Paragon nodes

The implementation of the *virtual Paragon node* concept includes the controlling process on the *virtual service partition* as well as the application processes on the *virtual compute partition*. In the following we will first introduce how the VPN concept has been implemented on the *virtual compute partition* and then discuss the implementation on the *virtual service partition*.

The goal of *virtual Paragon node*s is to have an equivalent to a Paragon node which consists of the node hardware, the operating system on that node and the application processes on that node. A natural approach to model this environment is to introduce a daemon process which is responsible to virtualize the node hardware and the operating system. The application processes' calls to NX communication routines are transformed into requests to the the daemon process. Like on a Paragon system the application processes are clients which request some service from the operating system. But in difference to that every system call would require an interprocess communication in such an implementation.

As an enhancement of the above described implementation we have introduced the following improvement which reduces the amount of interprocess communication. As not all system calls require the assistance of a centralized operating system parts of the operating system's tasks have been migrated into the application processes. Figure 5 shows a *virtual*

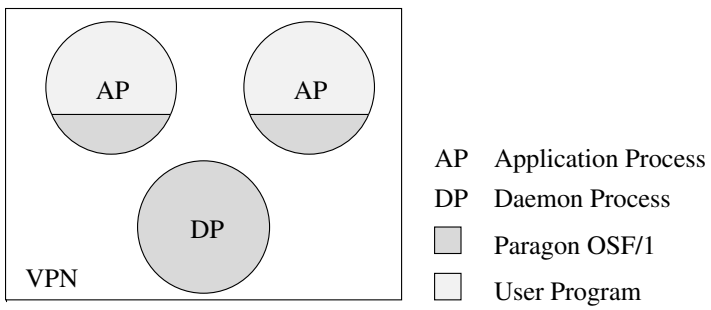

Figure 5: Processes and the distribution of the operating system on a VPN

Paragon node with two application processes and their corresponding daemon. Operations which can be carried out without the assistance of a centralized operating system must be independent of each other and must not address common operating system structures or

tables. These are for example NX send operations, as only address lookups of the destination address and a transformation to interprocess communication calls are necessary[2]. In contrast to that, changing the *ptype* of a process will require the daemon's assistance as only one process on a *virtual Paragon node* is allowed two have a certain *ptype*. The daemon as the central control instance to grant *ptype*s can easily guarantee their uniqueness on a single *virtual Paragon node*.

The implementation of the VPN on the *virtual service partition* is different from the approach described above for the *virtual compute partition*. But as the tasks of the controlling process on the *virtual service partition* are different to those of the application processes on the *virtual compute partition* this difference is no contradiction to a uniform implementation. In contrast to the application processes where mainly computational work is done the controlling process has the following jobs: starting the application, managing the processes, propagating signals, providing I/O facilities and terminating an application. If a similar implementation had been chosen like for VPNs on the *virtual compute partition* frequent interprocess communication between the controlling process and its daemon would have been necessary. In addition there is only one process on the *virtual service partition*, so a natural improvement is to join the controlling process with its daemon into one process. For applications which were linked with the –lnx linking option the controlling process may also take part in the computation. This functionality is not affected by the decision to have only one process to implement the VPN on the *virtual service partition*.

4.2 Implementing the layers of NXLIB

For the following discussion concerning the implementation of the layer in the NXLIB refer again to figure 4. To reduce the effort which was necessary to implement and test the NXLIB the decision was made to use a communication protocol which supports both local and remote communication. For that reason we have chosen TCP-sockets which also offer a reliable point-to-point communication. Consequently no additional code was necessary to achieve a reliable communication protocol. Nevertheless the distinction between local and remote communication has been made throughout the whole implementation: the local and remote communication layer simply call the same basic communication functions. An exchange of the communication protocol in later versions is no problem as only the calls in either the local or the remote layer have to be substituted.

TCP-sockets are addressed via a descriptor which is similar to a file descriptor. The basis of the address conversion layer forms a table where all necessary information about the processes is stored. Functions to add, delete, update and retrieve this information are provided by this layer. The address mapping information which is necessary to communicate between different processes can be extracted from retrieved process descriptors[3].

The reliable communication layer uses the calls of the address conversion layer to retrieve information about a destination process of a send or receive call. Based on this information it issues the corresponding local or remote communication calls with the appropriate communication protocol addresses. For a further discussion of the implementation of the NX communication calls refer to section 4.3.

To handle incoming messages the buffer management keeps a message table where they are stored. During a send call a message type is associated with the message. The destination of the message is a process on a VPN with a dedicated *ptype*. A receive call on that VPN matches an incoming message if the current *ptype* of the process is the same as the one specified in the send call and if the message type is identical. Hence, the buffer management provides a set of calls to insert, retrieve and delete messages from the message table.

[2] This is basically always true for our implementation. Nevertheless under certain conditions a NX send operation will also require the daemon process. For a discussion of that case refer to section 4.3.

[3] Section 4.3 will show that on one process only information about such processes is stored, to which a communication path exists.

Depending on the fact whether or not a corresponding receive call was already placed and the type of the receive call, the Paragon OSF/1 communication layer invokes different actions. If a user specified message handler has been installed with a previous hrecv call, the handler is invoked and the message is deleted from the table. If on the other hand a synchronous receive was placed before, the message is extracted and the call returns with it as a result. A previous call to a asynchronous irecv call simply leaves the message in the table and marks it as received, so that later calls to probe functions can determine that the message is now available. In the case that no matching receive has been called at all the message is also inserted in the table until a later receive operation deletes the message from the table.

4.3 NX message passing calls

Section 4.2 already explained the functionality of the different NXLIB layers. This section provides an overview how these layers cooperate to simulate the Paragon message-passing calls on a network of workstations. Therefore, the following topics will be addressed: first the basic concepts are presented, then the start of an application will be described and the address resolution protocol will be explained.

4.3.1 Implementation concepts of NX message-passing calls

An important issue for message-passing programming libraries is the latency of the communication calls. 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 are necessary until a message is sent. On the other hand, on most UNIX systems the descriptors which are available for open files and sockets are limited. A full interconnection of all application processes would therefore reduce the number of processes in an application drastically. 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. 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. And 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.

4.3.2 Start of an NXLIB application

Like on a Paragon system an application is automatically started if it was linked with the –nx linker option. If the –lnx switch was used the programmer is responsible to call the corresponding system calls in the controlling process. As the basic sequence of system calls is the same the following describes only the –nx case.

To start the application the user simply types the name of the application at a command line prompt. The command is started as any conventional UNIX command and executes a

nx_initve call. This call reads the partition mapping file and, as illustrated in figure 6, starts the daemons of the required VPN on the specified machines. The next step is the invocation of the nx_loadve call which initiates the creation of the application processes.

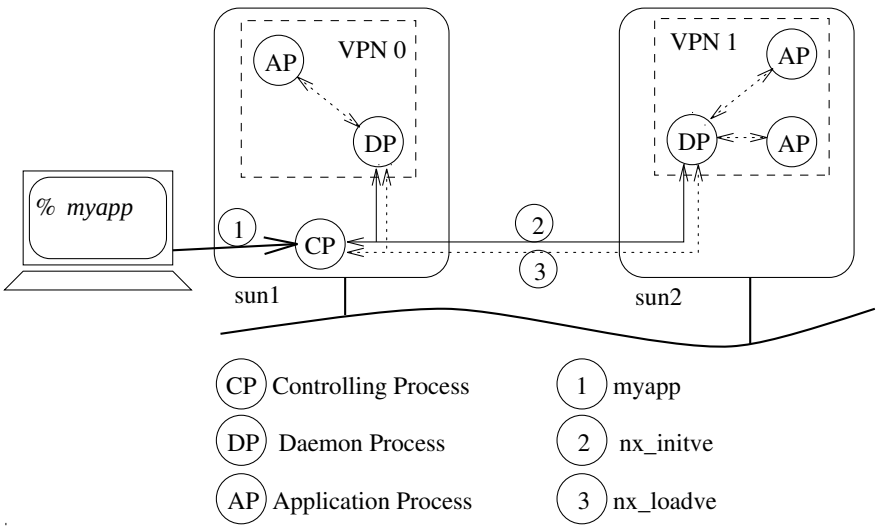

Figure 6: Starting an NXLIB application

The daemons on the remote machines are currently started via a standard Berkeley rsh command. The daemons on the remote machines inherit the environment of the machine where the application was started. A prerequisite for starting the node program is that the binary on each workstation is located somewhere in the PATH environment variable of the machine where the controlling process is located.

4.3.3 Address resolution protocol

Concerning communication links with TCP sockets the situation after the start of an application is as shown in figure 7: the daemon processes are connected to each other and the application processes are linked to their corresponding daemon. The address conversion

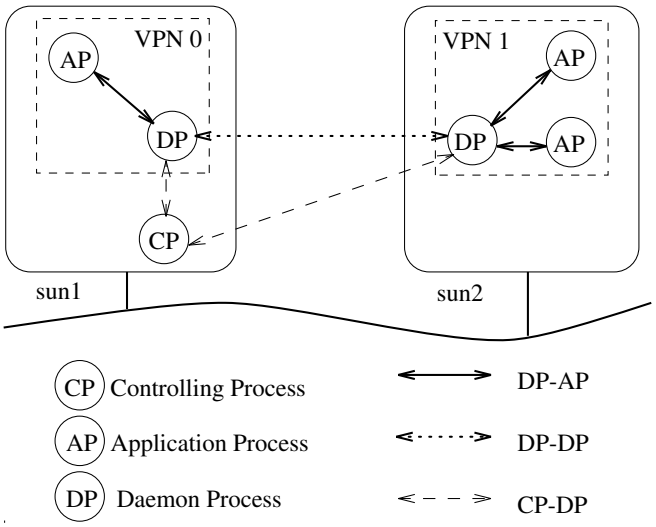

Figure 7: Configuration after starting an NXLIB application

layer within the daemons has information about all other daemons and the application

processes of its associated VPN. The application processes on the other hand have only address information about their daemon. During an application is executing further connections between application processes are created on demand when two application processes are communicating for the first time.

If an application process tries to send a message to a VPN to which no connection exists, its address conversion layer cannot retrieve a process descriptor for this process. To get the information of the requested process the address conversion layer contacts its daemon process with an ADR[4] protocol unit. If the daemon can provide the requested information it forwards it to the process with a DAA[5] protocol unit. Otherwise the daemon contacts the daemon of the specified VPN with a DDR[6] protocol unit. As this daemon is responsible for the VPN where the destination application process resides the necessary addresses must be stored there. Otherwise the application process does not yet exist and an error has occurred in the program. The daemon returns the addresses with a DDA[7] to the requesting daemon, which in turn updates its address conversion information and finally forwards the address to the application process with a DAA unit. In the last step the application process now contacts the destination process with a AAR[8] to establish a new socket connection. This results in a new point-to-point connection between the two processes which will be used for all further messages sent between the two application processes.

4.4 Implementing global operations

On a Paragon global operations manipulate data which are distributed among the nodes of an application, e.g. it is possible to calculate the sum of an array which is spread over the nodes. This requires the collection of data from every node. On a Paragon algorithms using a minimal spanning tree communication structure are used to collect the data. The same implementation could be used for NXLIB but as a network of workstations is coupled via an Ethernet bus the messages are serialized anyway. Consequently a similar optimization is not possible for global operations in NXLIB.

The implementation for global calls in NXLIB uses a simpler approach which is not less efficient on a network of workstations. As the execution of a global operation synchronizes the application processes the controlling process is used to collect, evaluate and distribute the result of a global operation. Therefore, all processes send a protocol unit via their daemon to the controlling process with the necessary parameters to carry out the operation. After that the application processes wait until the answer from the controlling process arrives. The controlling process on the other hand collects the incoming protocol units from every VPN, then computes the requested global operation and finally forwards the result of the computation to all application processes.

4.5 Workstation specific changes and restrictions

Although a network of coupled workstations basically has the same type of architecture as a multiprocessor system like the Paragon there are differences which put several restrictions on the implementation of NXLIB. A short summary of these restrictions and changes will be given in this section.

The compiler and linker on a Paragon system use special switches (–nx or –lnx) to create parallel applications. Compilers and linkers on workstations do not have an equivalent switch. To support an easy to use compilation system for Paragon applications which should run with NXLIB two special shell scripts have been provided. One to compile and link C

[4] ADR: application-daemon-request, address
[5] DAA: daemon-application-answer, address
[6] DDR: daemon-daemon-request, address
[7] DDA: daemon-daemon-answer, address
[8] AAR: application-application-request, message

applications and one for Fortran applications. These scripts can be called with the same parameters and options as the Paragon compiler and linker.

In contrast to the Paragon where a distributed operating system is used the workstations all have independent operating systems. Thus, the single system image which is provided on a Paragon is not fully available within NXLIB, e.g. the process identifiers are not transparent to every node, gang scheduling and priorities are not supported. Modifications of the operating system on every machine would have been necessary, to implement these features.

Due to different hardware units, features like the partition management and the configuration of the buffers used for messages have been neglected or are available in a different manner. An implementation of these features would require similar hardware as on a Paragon.

Finally several facilities of a Paragon system were left out during the implementation as only limited man power was available. These are the reactive kernel interface, the iPSC/860 compatibility calls, support for parallel I/O and the nx_nfork call.

5 Conclusion

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 multiprocessor 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.

With the exceptions mentioned in section 4.5 NXLIB offers the same software 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.

Due to the layered design the NXLIB can be easily extended and ported to further UNIX machines. Currently only Sun workstations are supported.

6 Future Work

Issues for further work were partially already mentioned in section 4.5. The improvement of the single system image which is available on a Paragon with its transparent process identifiers on all nodes, the gang scheduling facility and the priorities of processes in NXLIB would require modifications to the basic operating system of the workstations. As program sources are not available future projects will not cover these topics.

More important 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 stripping 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 during 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] M. Flynn. Very High Speed Computing Systems. In *Proceedings IEEE*, volume 54(12), pages 1901–1909. IEEE, 1966.

[2] Intel Supercomputer System's Division, 15201 N.W. Greenbrier Parkway, Beaverton, OR 97006. *Paragon OSF/1 C Commands Reference Manual*, 1 edition, April 1993.

[3] 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.

[4] Intel Supercomputer System's Division, 15201 N.W. Greenbrier Parkway, Beaverton, OR 97006. *Paragon OSF/1 Fortran System Calls Reference Manual*, 1 edition, April 1993.

[5] Intel Supercomputer System's Division, 15201 N.W. Greenbrier Parkway, Beaverton, OR 97006. *Paragon OSF/1 User's Guide*, 1 edition, April 1993.

[6] Neal Margulis. *i860 Microprocessor Architecture*. McGraw-Hill, Berkley, 1990.

[7] L.M. Ni and P.K. McKinley. A survey of wormhole routing techniques in direct networks. *IEEE Computer*, pages 62–76, February 1993.

[8] Open Software Foundation, 11 Cambridge Center, Cambridge, MA 02142. *The Design of the OSF/1 Operating System*, 1.1 edition, May 1992.

[9] Paul Pierce. The NX/2 Operating System. In *Proceedings of the 3rd Conference on Hypercube Concurrent Computers and Applications*, pages 384–391. ACM, 1988.

Acknowledgement

We would like to thank Intel ESDC, especially Dr. habil. Thomas Bemmerl and Bernhard Ries, for their cooperation. During all phases of the project they provided us with detailed information material and established helpful discussions with Intel SSD.