CONF-9706118--1

The Seamless Computing Environment

By

Timothy J. Sheehan* (sheehan@ccs.ornl.gov), Robert Pennington** (penningt@psc.edu),
Philip M. Papadopoulos* (phil@msr.epm.ornl.gov), George A. Geist (geistgaii@ornl.gov),
Richard Alexander* (alexar@ccs.ornl.gov)

** Oak Ridge National Laboratory / P.O. Box 2008 / Oak Ridge, TN 37831*

*** Pittsburgh Supercomputing Center / 4400 Fifth Avenue / Pittsburgh, PA 15213*

1 Introduction

Oak Ridge National Laboratory (ORNL), Sandia National Laboratories (SNL), and Pittsburgh Supercomputing Center (PSC) are in the midst of a project through which their supercomputers are linked via high speed networks. The goal of this project is to solve national security and scientific problems too large to run on a single machine. This project, as well as the desire to maximize the use of high performance computing systems, has provided the impetus to develop and implement software tools and infrastructure to automate the tasks associated with running codes on one or more heterogeneous machines from a geographically distributed pool.

The ultimate goal of this effort is the Seamless Computing Environment (SCE). SCE is a production environment to which a user submits a job and receives results without having to worry about scheduling resources or even which resources the system uses. The compilation, data migration, scheduling, and execution will take place with minimal user intervention.

We are using a twofold strategy to implement SCE. First of all, the parts of the system we implement will be usable at the time of their implementation. This way, users of the linked system will be able to be more productive as the overall system evolves, and they will be able to help steer development so that SCE serves their needs. Secondly, whenever possible, we will adapt and use existing technologies. Our goal is to assemble a system that enables scientists produce results, and our intention is to meet that goal as quickly and efficiently as possible.

This paper discusses various aspects of SCE and the linked computing project. Section 2 deals with the computers and networks used. Section 3 describes the strategies used in developing and debugging both applications and infrastructure. In section 4, management of linked machines is discussed. Section 5 discusses intermachine message passing. The SCE functions of scheduling and resource allocation are described in section 6. Section 7 discusses I/O and data migration. Section 8 deals with security. And section 9 draws conclusions.

2 Computers and Networks

DISCLAIMER

2.1 Introduction

Obviously, the two basic components necessary for SCE are hardware and networks. From the three labs currently involved in the collaboration we have amassed a heterogeneous collection of machines suitable for both testing and production. High-speed network connections between sites are in place and increases in bandwidths are scheduled. This section describes the hardware and networks used in the project.

2.2 Computers

ORNL has a total of four Paragon MPP computers involved in the experiment: a 1024 MP node XP/S 150, a 512 GP node XP/S 35, the 128 GP node XP/S 10, and an 8 (4 GP + 4 MP) node machine known as mpdot. The XP/S 10, 35, and 150 run the OSF/1 operating system, provided by Intel, full time and are used for development and production by general users. The mpdot is a test machine and runs a mix of OSF/1 and Puma, an operating system developed by SNL for Paragons and the TFLOPS machine. The XP/S 150 and 35 are both equipped with GigaNet ATM OC-12c Protocall Engine interfaces.

SNL has two Paragons involved in the experiment: an 1840 GP node XP/S 140 and a 56 node XP/S 4. The XP/S 140 is a production machine and runs the SUNMOS operating system nearly full time. For linked computing runs, the XP/S 140 is booted with Puma. This is necessary because SUNMOS does not allow compute node tasks to communicate with service node tasks, thus precluding intermachine communication. Puma has this functionality. The XP/S 4 is used primarily for development and runs a mix of OSF/1, SUNMOS and Puma. Both of these machines are equipped with GigaNet ATM OC-12c Protocall Engine interfaces.

PSC has a 512 PE T3E, with 256 PEs running at 300 MHz and 256 running at 450 MHz. The T3E runs the UNICOS operating system. Currently, PSC's T3E has no native ATM interface but possibilities for ATM connectivity include an external ATM converter and an internal ATM interface.

2.3 Network Connections

With a single GigaNet ATM board installed in a Paragon, the potential intermachine bandwidth is OC-12 (655 Mb/s), and between machines at the same site, both ORNL and SNL have transmitted data at that rate going through an ATM switch. In addition, since the two large ORNL machines are located in the same computer room, they can be easily connected by a direct fiber link, bypassing the ATM switch.

Intersite ATM connectivity between ORNL and SNL is via the DOE ESnet. ORNL's connection is OC-3. SNL's connection is currently DS-3 with an upgrade to OC-3 scheduled in June, 1997. PSC has OC-3 ATM connectivity via vBNS with plans to upgrade to OC-12 by the end of June, 1997. For either ORNL or SNL to connect into PSC, they must go through an internetwork cross connect. An OC-3 cross connect between ESnet and vBNS is expected in the near future. However, until the T3E has ATM connectivity, traffic between them and either of the other sites will utilize IP over ATM.

3 Development and Debugging

3.1 Introduction

Development of the SCE infrastructure and applications that utilize it have proceeded in tandem. While this method of co-development provides additional challenges, it also insures immediate user input to SCE system developers. This section discusses issues and procedures for application code development with the SCE, strategies utilized for infrastructure development, and debugging strategies for applications utilizing the environment.

3.2 Application Development

The first step in developing an application for any environment is the selection of an appropriate algorithm. For applications which utilize only one machine from the pool, this is straightforward. The machine used by the code must have the processing power and adequate memory to solve the problem. Right now, all resources in the pool are MPPs. As different architectures, for example large shared memory machines and clusters of SMPs, become available in the environment, certain algorithms may be appropriate for only a subset of machines. Alternatively, different algorithms could be specified at compile time depending on the target architecture.

For linked machine runs, algorithm choice becomes more complex. Intermachine bandwidth is lower than intramachine bandwidth. While this problem will be continually mitigated by higher performance networks, it will never go away completely. Intermachine latency, on the other hand, is a problem with hard limits. Once the most direct connection between sites is made, the limiting factor is the speed of light. Algorithms chosen for linked runs must be able to sufficiently overlap computation with communication in order to make the gain in aggregate performance worthwhile.

An additional issue for intermachine runs is load balancing. In an environment of machines performing at different levels, the algorithm should be able to shift more of the work to the higher performing machine. Memory considerations come into play with this strategy. If the amount of memory associated with the greater amount of work is not available on the higher performing machine, then the problem will not be able to be distributed in a balanced fashion. For memory bound problems a user may find that sacrificing load balancing is a small price to pay in order to solve a problem that may be insoluble otherwise.

Once algorithm selection is complete, the code must be implemented on each appropriate machine in the environment. Codes are scaled up, graduating to larger production machines as the limits of the test environment are reached. Linked code development proceeds similarly beginning with small test machines linked via TCP/IP.

3.3 Infrastructure Development

The testing and scaling philosophy used for application development is also applied to the development of the infrastructure. As components are added, whether network links, machines, or software, they are tested alone and then in combination with small parts of the system before being tested with larger parts and the whole.

The ATM switches provide an example of this. ORNL and SNL got their ATM switches at about the same time. Each site first did tests using loopbacks from their small computers. The switches were then tested between local machines. Testing proceeded with loopbacks to each other's site, then between test machines. Only when a high level of confidence in the switches, and the cross country network, were successfully completed did testing between production machines begin.

3.4 Debugging

When working on a single, stable machine, a bug that manifests itself during the running of a code is relatively straightforward to fix. Problems are most often due to the application, and debugging normally proceeds with that assumption. The user starts the process by isolating the point in the code where the problem occurs.

In the linked environment the system is much more complex. An intermachine message layer, currently PVM, which interacts with the operating system is added, as is a myriad of hardware including network interface boards, switches and routers, and fiber. While initial testing of these hardware and software components is performed before codes utilize them, each application stresses the system in a different way. More often than not, when a problem occurs it is not simply an application bug. Isolation can prove difficult. The strategy that has proven most efficient is one in which we go back to a simpler configuration under which the code runs reliably and then add components of the system to determine what single change in the system causes the problem.

For example, when running a particular application between the two Paragons in the ORNL computer room, all of the packets going from one machine to the other were lost. Testing of the code using IP over Ethernet instead of AAL5 over ATM gave us confidence that the problem was not a code problem or a problem with the part of PVM that does not use ATM. As a next test, we connected the machines using direct fiber instead of going through the ATM switch. The code still ran successfully. The problem manifested itself again when the ATM switch was included in the intermachine connection. At that point we concluded that the problem was due either to the switch or an interaction between the switch and the Paragon's ATM interface. Test codes isolated the problem as a switch problem.

The fact that application codes evolve, though, always raises the suspicion that an observed problem is due to a change even when the change is well tested. And in reality, this is sometimes the case. In order to streamline the debugging process, we are collecting a set of well understood, stable application and benchmark codes that can be used to test the system when part of the infrastructure changes or when another application runs into a problem. In doing this, we will remove a significant variable from the debugging process.

4 Managing Linked Distributed Machines

Our distributed computing testbed is composed of a heterogeneous mixture of supercomputers, networks, and operating systems. The environment includes two types of networking technologies (AAL5 over ATM and TCP/IP over ethernet), two chip architectures (Intel I860, Dec Alpha) and three operating systems (OSF/1, Puma, and Unicos). To keep this heterogeneity manageable, an abstraction layer is needed to perform message passing, parallel I/O, and virtual machine control.

PVM was chosen as the abstraction layer because:

- It already contained the necessary virtual machine control features.
- It provided interoperability between different vendors.
- It had already been used to connect widely separated supercomputer sites.
- It could easily be modified to handle the ATM connection.

In addition, all the initial applications in this environment were developed to use PVM message passing

calls. (It is planned that the message passing interface API (MPI) [1] will also be supported when applications appear that need this API.)

PVM is implemented around the concept of daemons and tasks. A single PVM daemon (pvmd) runs on each host in the collection. A collection of communicating pvmds makes up the user's virtual machine. The daemons are installed as user-level processes. Multiple users may overlap virtual machines without compromising security or message integrity. Pvmd's route messages, hold globally available information like machine configuration, and manage tasks (spawning, signaling, exit notification). It is the feature of process and machine control that really differentiates PVM from a pure message passing interface like MPI. While MPI and PVM overlap in terms of basic messaging, MPI has a much richer set of communication primitives, whereas PVM supports processes inquiring and dynamically manipulating the configuration of the virtual machine. PVM and MPI overlap functionality, but neither is a proper subset of the other. In fact, when MPI is added as a supported message passing library, virtual machine control must still be handled by some other method such as PVM.

PVM is used for creating and controlling linked distributed computers and implements a multi-protocol messaging layer to provide a familiar API to the programmer. Several important modifications were made to PVM to include the ability to operate through firewalls, utilize ATM inter-machine connections, and support the Puma OS. The basic structure of PVM provides a single daemon per machine, the pvmd, which runs in the service partition of each Paragon. Task-to-task messages that are to stay on a particular machine, bypass the pvmd and use the underlying scalable (NX, in this case) MPP message layer.

Because of local site security policies, a virtual machine cannot be brought together without first using TCP/IP-based authentication. The security policies also disabled all remote execution commands like rsh and rexec that PVM typically uses for virtual machine configuration. Thus to configure a parallel virtual machine the user must log into each of the machines using s/key one-time passwords and manually start the pvmd. This means that daemons are configured initially to use TCP/IP as the intermachine transport.

By giving a single command, AAL5 then replaces TCP/IP transport. This configuration command specifies the machines that should connect together on a particular virtual channel (VC). With this architecture, the Paragon daemons become multi-protocol routers that switch messages among sockets (UDP, TCP), ATM, and native communication. Sockets are used to talk to non-ATM capable machines and to local service node tasks. Native NX communication is used by the daemon to talk to its local tasks on the compute partition. This protocol switching requires the daemon to enter a busy-wait polling loop because the three protocols cannot be multiplexed in a single blocking wait call. The AAL5-capable daemons maintain interoperability with standard PVM and keep true to the idea that messages should travel over the fastest available transport.

Further enhancements to SCE will make virtual machine use easier. The requirement of logging in to each machine for security purposes will be dropped (see Section 9 Security). Changes to PVM that will allow automatic virtual machine configuration and application start up are planned. Users doing development as well as the schedule and resource allocation system (see Section 6 Scheduling and Resource Allocation) will be able to take advantage of these features. Also, enhancements to PVM for fault detection and reaction will be added. This will include the ability for PVM to detect the failure of part of the virtual machine and to shut down the rest of it so that resources are freed for other jobs.

5 Message Passing Between Supercomputers

In this section we will discuss some of the modifications made to PVM to support low-level ATM protocols and the Puma OS for message passing. From the outset, the modifications made to the PVM runtime system were designed so that no changes to the application's source were necessary. All modifications to enable messages to travel over low-level ATM are performed in the runtime system rather than as a compile-time option. This allowed for simple testing of the code over TCP/IP and AAL5 without re-compilation.

To understand the modification made to PVM, one must first examine the pathway that a message takes in the PVM system. From a task's point of view it only sees PVM message-passing calls, but underneath PVM translates these into NX messages in the case of Paragons or native T3E communication on the Cray. If a message is bound for a destination that is on-machine, the message is sent directly to the task, bypassing the pvmd. If, on the other hand, the message is destined off-machine, the data is first sent to the local pvmd, forwarded to the remote pvmd and finally delivered to the remote task. This task--pvmd--pvmd--task routing is called a three way route.

The advantage of a three-way route is that all of the complexity of creating an intermachine connection, providing message sequencing and reliability over that connection, and monitoring it for remote host failures can be put in the daemon instead of each task. It also allows us to multiplex the intermachine connection among all tasks using fast native messaging to go between tasks and daemons. A disadvantage of this routing is that the extra hops add more latency to the network connection, making this route a performance bottleneck. Our modifications were to replace the pvmd--pvmd connection with a AAL-5 encapsulated ATM connection. This reduced latency significantly by bypassing the IP stack and dramatically improved bandwidth. It should be noted that AAL-5 encapsulation is an unreliable protocol, and the daemons are required to guarantee message delivery through a timeout and retry scheme.

A significant issue to contend with in the message routing is getting messages to flow between the multiple partitions within an MPP. The Paragon operating system divides the single mesh of CPUs into three partition types: compute, service, and I/O. On all of the Paragons in our testbed, OSF/1 must run on the service nodes. For Puma-based machines (Sandia), the lightweight Puma kernel runs in the compute partition, while the OSF/1 is used for the pure OSF machines (Oak Ridge). Compute-node tasks compiled for Puma are loaded and monitored by a special OSF/1 service node task called a yod. In early distributions, yod had an interface to perform communication with the compute-node tasks, but it was not exported to user programs. As part of this project, a communication library was written by the Puma team to enable the service task (OSF/1) to task (Puma) communication. This library has been used by other groups to build debuggers and task managers for Puma. It should be noted that the Giganet ATM cards only provide drivers that run under OSF. So, ATM-bound messages must first go through an OSF node before being transferred off-machine.

Currently, the route of any off-machine message on the Paragon includes the service node. This has proven to be a bottleneck limiting intermachine PVM over ATM throughput to approximately 17 MB/s. Work is underway to send ATM messages to the ATM node going through one or more nodes in the compute partition instead of through the service node. We anticipate that this direct-to-router messaging will yield data throughput of approximately 72 MB/s.

The T3E provides a similar service/compute partition scheme except that the service "partition" is an entirely different machine and architecture. No native ATM interface is currently available for the T3E. A very real performance problem exists to enable fast communication between Paragons and the T3E.

The Paragon has a very slow TCP/IP stack and is unable to saturate a 10 Mbit ethernet link. For efficiency, the IP stack of the machine must either be bypassed or replaced. The T3E can manage reasonably fast IP sessions (several megabytes/sec) but would be throttled when communicating with a Paragon.

The intermachine communication using AAL5 was integrated into PVM in the following way. The standard PVM daemon-to-daemon communication utilizes UDP as its basic transport. For this testbed, UDP is replaced by AAL-5 packet encapsulation which, although unreliable like UDP, is sequenced. For simplicity, the same PVM reliability algorithms are used for both AAL5 and UDP. AAL5 can be "switched-on" at runtime independently of any application. The GigaNet API [2] allows any CPU to bind to a particular virtual circuit identifier (VCI) and communicate to the "other end" over this VCI. The testbed currently utilizes permanent virtual circuits (PVCs) because no provision is available in either the OC-12 switch or in the ESnet cloud for switched virtual circuits (SVCs). Essentially, virtual circuits are hardwired in the network switches and PVM must communicate using these hand-configured connections.

6. Scheduling and Resource Allocation

A scheduling or resource allocation system is necessary for the SCE to provide the level of functionality necessary for an application to be initiated and successfully executed across the three supercomputing sites on a regular, production basis. The basic requirements for such a scheduling system are:

- Monitoring of systems for availability and state
- Allocation of resources
- Reservation of resources at each site
- Interface to existing schedulers - NQS and NQE
- Job initiation and monitoring across the systems
- Reclamation of resources from the application and clean-up

As part of this work, we expect to develop a metascheduler that is able to perform the following functions:

- Monitor resource availability on heterogeneous machines and networks
- Monitor application performance on these machines and networks
- Interact with the schedulers at each of the sites to provide access through the site's native batch scheduler, or directly for interactive running
- Initiate and control PVM applications in a secure manner

This scheduler will not replace the schedulers at the individual sites. It will collaborate with them to correctly place the application on the machines. This will require some administrative work at each site to configure the native batch scheduler to allow such use.

The system design will include a low overhead daemons that are able to monitor the heterogeneous systems at the different sites and interconnecting networks that comprise the the total system. The daemons must be able to efficiently and unobtrusively monitor the state of the systems, networks and the performance of the application and should offer usable APIs to the scheduler and the application.

The monitoring will include information on the relevant (although machine dependent) parameters of

the systems, including the obvious ones such as status and availability of system resources (CPU resources, memory and temporary storage on primary filesystems), status and availability of essential intra-site resources (secondary/archival storage systems, local high speed networks) and inter-site resources (primarily network performance and status). The scheduler will be able to obtain this information for scheduling decisions and the application will able to use it for load distribution, either statically at startup or dynamically during execution.

The initiation of the virtual machine on which the application runs will also be performed by the scheduling system. The metascheduler must necessarily be designed and constructed with sufficient security measures to be trusted by each site. This trust can be used to set up the virtual machine on behalf of an authorized user. Providing this service at an administratively higher level than that of the individual user should help minimize the security concerns of the sites as well as provide a reasonable level of assurance to the user that the application will be launched successfully. In addition, by allowing the metascheduler to retain a measure of control or contact with the PVM application, it should be possible for it to act appropriately in response to normal or abnormal termination of the application or to difficulties (identified by the monitoring system) with the systems that make up the virtual machine.

The current method of initiating an application that includes multiple sites is to manually coordinate the availability of the resources with site administrators and then initiate the PVM based application by hand. We have developed a work plan to implement an automated version that will successively reduce the necessity of interacting with each site for an application run. We are also tracking developments by Platform Computing with their Multi-Cluster work (http://www.platform.com) and the work of the informal Psched group to develop APIs for support of parallel processing and job schedulers (http://www.lovelace.nas.nasa.gov/Psched/Psched.html). The final result of this work will be the deployment of an infrastructure that will allow the application user at one site to make a resource request that can be honored and executed across the sites in the SCE.

7 I/O and Data Migration

7.1 Introduction

Ultimately, from the standpoint of the user, three main factors determine the effectiveness of a file system: reliability, ease of use, and performance. The underlying implementation of the file system is unimportant if it meets these goals while providing full functionality. At run time, an unreliable system causes the run to fail. A slow system wastes cycles while the application waits. If the file system is difficult to use, for example requiring machine dependent APIs, code development and testing time increases.

After a run completes, the file system still needs to fulfill these goals. Archiving and retrieval should proceed smoothly. If a file is to be used on a single system for post-processing, the format of the data should be consistent (i.e. the same format at the binary level regardless of the machine on which the application was run). Access to stored data should be straightforward and proceed without imposing unreasonable delays.

If jobs will be migrated to one or more distributed machines, so must the files associated with them. Both the size and purpose of an individual file must be taken into consideration when designing the systems used for data storage and migration. This section discusses issues related to the types of files associated with running applications and then describes how SCE will handle them.

7.2 Source and Executable Files

A user normally maintains a single source tree for an application in order to simplify source code maintenance. When compiling for a run, the source is either transferred to the target platform or a cross compiler is used. For convenience, a user often keeps executables on the same file system as the source and migrates an executable via ftp prior to a run. A user should be able to avoid the tedious process of executable migration and management that often leads to the wrong executable being run.

7.3 Configuration and Log Files

Configuration files are normally small (tens to thousands of bytes) and contain data that are used to define parameters for a run. Normally, these files are read once at the beginning of a run. When running on a variety of machines, a user must migrate the configuration file to the machine where the code will run. Maintaining many configuration files distributed on a variety of machines can prove to be difficult, and mistakes can cause failed runs or wasted machine time.

Many applications also output a summary or log file so that the user can monitor the job while it is running or examine summary data afterwards to analyze how the run went. The user should be able to specify a single location for this file and to examine the data in it in real-time while a run progresses. This necessitates real-time output of the data to a single file in a specified location, no matter where the code is running.

7.4 Input and Output Data Files

Input and output files are normally large, often on the order of hundreds of megabytes or gigabytes. The performance of an application depends in part on how fast this data can be input and output. Migrating such large files real-time during a run would have an adverse impact on performance. From the standpoint of a running application, these files should reside on the local high-performance file system.

For linked runs, how a large file is distributed between multiple machines also becomes an issue. Nodes in a distributed application do not normally require the data output by other nodes, so during run-time a distributed output file will not affect performance. When this criterion is met, files do not need to be moved between machines in real-time. Intermachine transfer of most distributed files will be done either in the background or after a run completes.

Output files from linked runs may be needed for continuation of a run or for post-processing inside or outside SCE. The platforms using the files may vary from one run to another. Functionality to handle all these scenarios will be built into the system. Distributed files will be able to be condensed into a single file image in one location, and a single image file in one location will be able to be distributed.

Currently platform specific I/O calls are implemented at compile time. Files are converted to the appropriate format and ftped to each machine before the run. The data produced on each local machine is written to that machine. After a run is complete, the data is then transferred to a single platform, converted to a common format, and merged into a contiguous file image. This is normally done interactively using utilities written and modified to handle the particular distributed file.

7.5 Data Migration Strategies

For source, executable, configuration, and log files, DFS under DCE will provide a solution. Each platform will see a single file system. Source files can be maintained in a single file system visible to the compiler on each machine, making source code migration unnecessary. The executable can be run from DFS or, SCE will move it to the target platform with a simple copy. Configuration and log files can be read from and written wherever the user desires in DFS.

Large data files present a more complex problem. The data is too great to move real-time, the format of binaries varies between platforms, and high performance i/o calls differ from platform to platform. A common API implemented for each platform in the environment will be used. Functionality relieving the user of managing the distributed nature of files will be included. Background and post-run data format conversion and transfer to archive will be included. For linked runs, whole files will to be constructed from their distributed parts before being archived. This work remains to be done, but we hope to make use of existing standards, such as the MPI-IO specification included in MPI-2.

Files will be archived to mass storage systems. Both ORNL and SNL have implemented HPSS systems. File transfer to and from HPSS will be via parallel ftp (pftp). PSC's archive system is managed by Cray DMF. Access to files in the archiving system is through ftp, a custom interface or DFS. The File ARchiver custom interface (FAR) provides secure, high performance remote access for the PSC supercomputing platforms. SCE will provide a common interface to allow access to these different systems.

8. Security

Security is a major concern with this project. SCE batch jobs, as well as users doing development work, will need access to the machines, file systems, and networks involved. While firewalls provide the desired level of security, they preclude easy access to parts of the overall system. This defeats the whole goal of the SCE. We have chosen the Distributed Computing Environment (DCE) to provide authentication and authorization for intersite access.

With DCE, two or more cells can enter into a trusted relationship. Requests between trusted cells are treated much like requests within a single cell. The formal creation of a trusted cell happens when a principal in each cell, that represents the other cell, is created. Once a user is authenticated within his home cell, he can request authorization to access the trusted (foreign) cell. Access is allowed because the request is coming from a trusted cell. Access extends both to computing resources and, through DFS, to files.

OSF/1 on the Paragons supports neither Kerberos, an authentication system that is embedded in DCE, nor DCE. We are testing MIT Kerberos 5 Version 1.0 as a client on the Paragons. We intend to use the DCE authentication services as the "server" for this client. This will enable a user to "kinit" to get credentials from the local DCE Authentication Service. Using the Kerberos cross-cell authentication function within DCE should allow a single job to be authenticated in multiple cells.

9 Conclusions

ORNL, SNL, and PSC have been successful in creating much of the infrastructure necessary to implement SCE and further development of this infrastructure continues. Each of the sites has a world class production MPP. Additional ORNL and SNL have testing environments in place. The high speed network connections are approaching bandwidths that will enable data to flow at the necessary rates for

linked production runs. PVM, the intermachine message passing layer being used, has been modified to take advantage of the the underlying hardware and network. Further modifications to PVM for increased bandwidth into and out of the ORNL and SNL Paragons is under way. Other enhancements will include automatic virtual machine configuration as well as fault detection and reaction.

Test runs of real applications have taken place on linked machines. Between ORNL and SNL, a 2048 node run of the CTH shock physics code has been achieved. Between ORNL and SNL, a 1024 node run of the LSMS first principles materials code has taken place. A 1372 production run of the LSMS has been done between ORNL's XP/S 150 and XP/S 35. These successes illustrate that a geographically distributed computing environment is a viable way, perhaps the only way, to run the largest applications. Our experiences in bringing about these successful runs have given us insight into the requirements and architecture of a seamless computing environment.

SCE continues to move towards realization. Several parts are either under development or have been implemented and will soon be integrated into the environment, for example, HPSS and DCE. Other parts, like the distributed file system API and the scheduling system, exist as high level designs and still require refinement and development. As we move forward, though, we will continue to implement SCE in usable increments, making users' lives easier at each step until all parts are in place.

References:

[1] W. Gropp, E. Lusk, A. Skjellum. Using MPI, MIT Press, Cambridge, MA, 1994.

[2] D.R. Follett, M.C. Gutierrez, R.F. Prohaska. A High Performance ATM Protocol Engine for the Intel Paragon, Available at http://www.cs.sandia.gov/ISUG/html/hspe.html, 1995.

==========

The authors acknowledge the use of: the Intel Paragon XP/S 5, XP/S 35, and XP/S 150 computers located in the Oak Ridge National Laboratory Center for Computational Sciences (CCS), funded by the Department of Energy's Mathematical, Information, and Computational Sciences (MICS) division of the Office of Computational Technology Research; the Intel Paragon XP/S 140 computer located at Sandia National Laboratories; and the Cray T3D and T3E computers located at the Pittsburgh Supercomputer Center.

This work is sponsored in part by the U.S. Department of Energy and the Center for Computational Sciences of the Oak Ridge National Laboratory under contract DE-AC05-96OR22464 with Lockheed Martin Energy Research Corporation.

This work was supported in part by the United States Department of Energy under Contract DE-AC04-94AL85000.