SANDIA REPORT
SAND2001-1784
Unlimited Release
Printed June 2001

Programming Paradigms for Massively Parallel Computers: LDRD Project Final Report

Ronald B. Brightwell

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865)576-8401
Facsimile: (865)576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.doe.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800)553-6847
Facsimile: (703)605-6900
E-Mail: orders@ntis.fedworld.gov
Online order: http://www.ntis.gov/ordering.htm

SAND2001–1784
Unlimited Release
Printed June 2001

Programming Paradigms for Massively Parallel Computers: LDRD Project Final Report

Ron Brightwell
Computational Sciences, Computer Sciences, and Mathematics Center
Sandia National Laboratories
P. O. Box 5800
Albuquerque, NM 87185-1110

Abstract

This technical report presents the initial proposal and renewal proposals for an LDRD project whose intended goal was to enable applications to take full advantage of the hardware available on Sandia's current and future massively parallel supercomputers by analyzing various ways of combining distributed-memory and shared-memory programming models. Despite Sandia's enormous success with distributed-memory parallel machines and the message-passing programming model, clusters of shared-memory processors appeared to be the massively parallel architecture of the future at the time this project was proposed. We had hoped to analyze various hybrid programming models for their effectiveness and characterize the types of application to which each model was well-suited. The report presents the initial research proposal and subsequent continuation proposals that highlight the proposed work and summarize the accomplishments.

Acknowledgment

Appreciation is extended to the following people for contributing to this research: Bill Hart, Steve Plimpton, Mark Sears, Mike Heroux, and Rolf Riesen. In particular, Bill Hart was the original principal investigator for this project and was responsible for much of the content of the initial proposal.

Contents

Programming Paradigms for Massively Parallel Computers: LDRD Project Final Report

1 Initial Proposal

The goal of this project is to enable applications to take full advantage of the hardware available on Sandia's current and future massively parallel supercomputers. Currently, nearly all Sandia codes on the Intel Paragon and ASCI Red [8] machine use only one of the two processors available on each node with only a single thread of execution. On ASCI Blue and the Computational Plant (Cplant™) [2] machines, this shortcoming will be exacerbated. More generally, the future of high-end computing appears to be clusters of SMPs, i.e. a mixture of distributed and shared memory hierarchies is exposed to the user. Learning how to program such machines effectively to achieve maximum impact (both inside and outside Sandia) with simulations is a must. The best model for each of the different types of applications will be discovered through implementation and exploration of several programming paradigms from both the system and application perspectives. More specifically, the following programming paradigms which are described in more detail in the body of this proposal will be examined:

1. Virtual nodes (VN)
   Each processor on a node is treated as a single node
2. Multiple virtual nodes (MVN)
   Simulation is decomposed into more nodes than physical processors
3. Single-node threads (SNT)
   One node contains multiple threads of execution running on only its processors
4. Multi-node threads (MNT)
   Threads of execution can migrate between nodes

For each of these models, (a) the user interface that the programmer sees will be defined, (b) the necessary system software to support the interface will be implemented, and (c) kernel computations will be modified or applications will be rewritten to conform to this new interface. These steps will provide the ability to contrast and compare the performance of various prototypical applications using the different programming models. The outcome of this effort will be two-fold:

a. It will provide insight into how to easily and effectively parallelize codes for clustered SMPs.

b. It will provide robust programming models for clustered SMP machines of the present and future and allow them to take full advantage of the hardware capabilities.

Note that it has become apparent that vendor support cannot be relied upon for (b) and that it must be done independently. Thus, this project fills a void that exists in the current hardware acquisition and software development plans.

There are five major goals for the first year:

(i) Survey parallelization methodologies that are appropriate for networks of SMPs.

(ii) Evaluate the feasibility of various parallelization methodologies.

(iii) Evaluate current thread implementations within Linux.

(iv) Develop preliminary implementation of threads that can support fine-grain message passing within Linux.

(v) Determine the basic requirements of the tools that will support the four programming paradigms.

1.1 Technical Problem

Sandia has enjoyed a long and fruitful history of extracting maximum performance from distributed-memory parallel machines with hundreds to thousands of nodes. Sandians have become experts at writing parallel applications in a particular programming model known as single-program multiple-data (SPMD) that is ideally suited to these kinds of machines. Almost all of Sandia's large-scale applications have been designed for this model with great success.

It is clear that massively parallel machines of the future will no longer have a purely distributed memory architecture. Single nodes of these machines will have many processors that share a large memory address space. The entire machine will expose a complex memory hierarchy to the user: multiple levels of cache on a processor, shared memory across a few processors, and possibly even distributed memory across nodes. Additionally, these machines introduce a communication hierarchy, since interprocessor communication differs between processors on a single node and processors on different nodes.

One of the greatest technical challenges that needs to be overcome is the development of programming models for this architecture that enable applications to achieve maximum performance. The SPMD programming model is not sufficiently powerful to encompass the more complex memory and communication hierarchy of these machines. For example, the options now provided by the operating systems on Sandia's Intel Paragon and the ASCI Red machine only provide SPMD-parallelism at the node level. Thus, almost all of Sandia's parallel codes only use one of the two processors on each node; half of the purchased compute power is essentially wasted. On the two ASCI Blue machines, there are many processors per node. This may be true for the nodes of the Computational Plant machine as well, at least in the out years. The compute potential of these machines will be woefully underutilized if applications are limited to the current programming model.

1.2 Technical Issues

The goal of this project is to develop one or more new programming paradigms that will enable full parallelization of applications for hybrid distributed/shared memory machines (e.g. clusters of SMPs). Specifically, the following four programming models will be implemented and tested.

Virtual Nodes (VN) On a machine with P nodes each with M processors, allow the user to specify at run-time that the application should execute on $P \times M$ "virtual" nodes, rather than the standard P nodes. Since SPMD codes are typically designed to run on an arbitrary number of nodes, this requires no additional coding by the user. However the OS must be modified substantially to enable this to occur transparently. In particular, message-passing must be allowed to occur between processors on a node, and a variety of OS system calls (e.g. I/O, malloc) must be modified to be called by multiple processors simultaneously. Multiple copies of the program's data will be stored in the node's shared memory.

Multiple Virtual Nodes (MVN) This model is somewhat analogous to the data-parallel model provided by the Connection Machine. If VN can be implemented, it should be possible to allow a physical processor to execute (and store) more than one "node" of the problem decomposition. Thus, the user could partition the problem into a large number of nodes N without even considering the number of processors P on which it will run. The OS would assign multiple virtual nodes to each physical processor. Like VN, the MVN option should be transparent to the application, requiring no code modifications.

Single-node threads (SNT) This is the traditional approach provided by vendors and the emerging POSIX standard for exploiting shared-memory parallelism on a single SMP cluster. However, implementing the POSIX standard on a cluster of SMPs within a node-level OS such as Puma [9] presents new challenges that will be addressed by this project. In particular, many key parts of the OS and its supported extensions will have to be re-coded to be made thread-safe – e.g. message-passing libraries, numerical and language libraries, I/O routines, etc. On the application side, there are also many issues that must be addressed to enable the programmer to code at both the message-passing and thread levels simultaneously.

Multi-node threads (MNT) The most general kind of thread implementation on a cluster of SMPs would allow for threads to communicate and migrate across nodes. This will be difficult to implement, but will allow the sophisticated user maximum control over how parallelism is exploited in the application. This will require substantial modification to the node-level OS, as well as exposing portions of the OS to the user.

Clearly, the paramount issue is performance. Each of these different models has factors that influence the ability to achieve the desired level of performance. Some of these factors are:

Memory The amount of memory needed to support each model is different. For example, VN and MVN may require duplicate copies of the operating system and executable image to exist in memory that is shared between processors, possibly wasting large amounts of memory. Each of these models may differ in the opportunity to take advantage of the features of the memory hierarchy. It may be easier to keep cache hit rates high for an application that is memory bandwidth intensive or synchronize memory accesses for processors in one model over another. The utility of demand-paged virtual memory may be exploitable by certain models.

Scheduling The ability to schedule threads of execution and provide the application with mechanisms to control the scheduling policy impacts performance. The capability to gang schedule heavyweight processes across a set of processors for VN is desirable. Similarly, for SNT, it is desirable to have the application determine when threads of execution should obtain compute resources.

Application Interface The current model of MPI [6] may be insufficient for some of these models. While a mixture of threads and message passing might be a good approach, current research has exposed some limitations. Message passing implementations may not only need to be made thread-safe, but also thread-aware. Explicit message passing may also not be the correct approach. There are other mechanisms, such as one sided communications or compiler directives, that may offer some distinct advantages.

Portability Ultimately the ability to achieve performance from any of these models is dependent upon the underlying hardware. An implementation of MNT is best suited to an SMP with greater than two processors. Conversely, an implementation of VN is better suited to an SMP with fewer than three processors. The ability to easily adapt an application from one model to another, or to move an application from one implementation of that model to another implementation is desirable. Obtaining portability while still maintaining performance is a critical issue.

Classification The characteristics of an application that influence performance will need to be discovered for each of the different models. Also, the boundaries associated with these characteristics (i.e., the definition of memory bandwidth intensive) will also help to determine the classes of applications suitable for each model.

Programming with multiple lightweight threads offers developers additional flexibility and modularity currently not available with single, heavyweight process programming models. The decomposition of software using multiple threads offers the possibility of greater utilization of parallel hardware by enabling processors to flexibly switch between tasks that need to wait for external resources or interthread communication. Finally, software reuse can be achieved with thread libraries that provide basic services to parallel computations, thereby facilitating the development of parallel software.

1.3 Technical Approach

For each of the programming models described in the previous section, the following tasks will be performed:

(a) The interface to the programming model that is exposed to the user will be defined. A good interface is critical if the model is to be widely used and appropriate for a wide range of applications.

(b) The system software will be modified to fully support the interface. Sandia is in a unique position to be able to control the system software due to experience in developing and designing the system software for many current and future parallel machines.

(c) Prototypical applications and kernels with which to test the programming model will be selected. Codes will be modified to use the defined programming interface. Planned applications which already exist in parallel SPMD form and with which the team members are already familiar include: optimization codes, molecular modeling and electronic structure codes, FFTs, and linear solvers. In order to determine the efficacy of these models, each of these applications will be implemented in more than one model. The criteria used to make this determination are performance, portability, ease of use.

1.4 Expected Results

The two most important outcomes of this project are as follows:

1. Applications programmers will be provided with the needed insight as to how to parallelize codes for clustered SMPs most effectively and easily.

2. Clustered SMP machines of the present and future, such as ASCI Red, ASCI Blue, and the Computational Plant, will have robust programming models in place that are fully supported by the OS. This will allow programmers to choose the most appropriate model(s) for an application and will allow for experimentation with different styles of parallelism.

1.5 Creativity and Innovation

This research will develop a suite of global optimization algorithms that will complement Sandia's local optimization suite OPT++. This research will also serve to extend the optimization capabilities of the SGOPT library. Although both OPT++ and SGOPT include global optimization algorithms, the algorithms in these libraries have been primarily applied to computationally expensive engineering science problems that preclude a truly global optimization. A focus on inexpensive to moderately expensive objective functions will enable the development and application of algorithms that address a fundamentally different class of problems.

The algorithmic issues addressed by this research will lead to innovative optimization algorithms that represent a practical trade-off between the degree of optimality achieved and speed. Although there is prior research that examines the algorithmic issues considered, research into global optimization is still in the early stages. Little is known about factors that affect the appropriate trade-off between global and local optimization, especially for adaptive global optimization sampling methods. Few stopping rules have been developed for global optimization algorithms, and most of those that have been developed have been developed for simple optimization methods like uniform random sampling. Global optimization has traditionally been applied to unconstrained optimization, and constrained global optimization problems have only recently begun to receive significant attention. Finally, parallelization is typically performed by parallelizing the objective function evaluations in a straightforward manner, which is a different type of parallelism than that which is proposed.

The system software issues addressed by this research may lead to new approaches to achieving high performance in multithreaded message passing environments. The previous operating systems developed at Sandia, SUNMOS [5] and Puma, have clearly addressed and overcome many of the issues affecting the ability of applications to achieve TeraFLOPS performance on distributed memory MPP systems. Similarly, this research may help to identify obstacles and solutions to scalability in other programming models.

The risks involved in developing new global optimization methods are somewhat mixed. It is often not difficult to tailor global optimization methods to effectively solve a particular application or closely related class of applications. Consequently, confidence in the ability to develop effective optimization methods for the defined application areas is high. It is generally much more difficult to devise global optimization algorithms that are both robust and efficient on a broad range of objective functions. Consequently, the proposal to develop global optimization algorithms with broad applicability is the risky component of this proposal. The trade-offs between global optimization algorithms can be difficult to quantify, especially since the level of accuracy needed to find near optimal solutions can affect the relative performance of different algorithms.

1.6 Programmatic Impact

This research has the potential for high impact on a wide variety of existing Sandia programs and throughout DOE because of the fundamental role that parallel computation has begun to play. Parallel software is utilized in a wide range of application domains at Sandia, including agent-based simulation, molecular dynamics, physics simulations, discrete optimization, simulations of chemically reacting flows, and materials aging.

The new programming models that will be developed will impact both software development on Sandia's existing MP computers as well as future MP computers within DOE. Although the message-passing programming model used for Sandia's Intel Paragon and the ASCI Red machine has provided an effective programming model for MP machines, it has proven insufficient to provide good utilization of even small-scale SMP nodes. With few exceptions, the second processor on the Sandia's Intel Paragon is not utilized for basic computations (it's typically treated as a communication co-processor), and a similar underutilization is expected for many applications that run on the ASCI Red machine.

In the short term, the software tools developed will provide support for advance programming methodologies at Sandia. The operating systems group at Sandia will be able to integrate these tools into the OS used for Cplant™ and can provide Intel with additional capabilities for the Cougar OS, which runs on the ASCI Red machine. It is expected that this investigation will lead to insights that would provide a basis for developing a common DOE standard for parallelization methodologies. The parallel machines at DOE labs are increasingly becoming a common resource of DOE, so this goal is quite important. Similarly, it is expected that experience with these tools will enable the ability to provide feedback to OS vendors (e.g. SGI) concerning the utility of various parallelization methodologies and the possible weaknesses of the OS standards being developed.

Finally, the evaluation of these software tools with specific applications will lead to immediate impact parallel codes for (a) discrete optimization, (b) molecular dynamics and (c) quantum electron structure prediction.

2 FY99 Renewal Proposal

2.1 FY 98 Accomplishments

The first year of this LDRD was intended to jumpstart Sandia's use of SMPs. During the past fiscal year Sandia has purchased large SMP boxes from DEC and SGI and has gained access to SMP IBM boxes at LLNL. We have worked to understand the nature and peculiarities of both the architecture and programming models for these machines. Further, we have surveyed parallelization methodologies that are appropriate for SMP clusters and developed conclusions concerning where our research efforts would be productively spent. Based on this survey, we have evaluated four programming models, and we have noted the need for a general programming environment within which users can tailor their application to a particular machines hardware. We have also evaluated thread implementations in Linux, and have identified a new scheduling method that seems appropriate for scientific computing applications. We are working on a thread-aware implementation of message passing interface (MPI) which can take advantage of the extended capabilities that threads provide. Finally, we have begun to explore how we can leverage the application programming interface and constructs of MPI to provide the functionality needed by a hybrid cluster environment. We have expanded the expertise by collaborating with SGI/Cray and U. of Maryland to provide hybrid SMP libraries, models, and applications codes. In conjunction with this work, we have given two panel presentations and four seminars with day long discussions.

As described in the project proposal, the primary end product of the project is a better understanding of effective programming models for clusters for SMPs. For FY98, the focus of the project was on an initial evaluation of existing programming models as well as development of thread support, which we expect to be necessary to implement some of the programming models we have identified.

We satisfied the first three milestones of the project by surveying and evaluating programming models for SMP clusters. Also, while surveying programming models, we investigated the manner in which threads are managed and scheduled.

Our survey and evaluation of programming models focused on models which would be most useful for SMP clusters that are contained in a dedicated compute partition. This model is the same design that is

used in ASCI Red and Cplant™, and it avoids issues relating to CPU resource contention which require additional work-arounds (e.g. gang scheduling).

A distinguishing feature of parallel programming models concerns whether data is communicated between processors using message-passing or shared-memory. Message-passing is the dominant programming method for distributed memory (DM) and non-uniform memory access (NUMA) architectures. Although OpenMP has emerged this year as a potential standard for shared-memory parallelism, it seems unlikely that this programming model can be used to develop software which scale to the large-scale ASCI applications that Sandia needs to solve. Shared-memory methods may, however, prove a valuable component of a hybrid programming model which utilizes message-passing between SMP nodes and shared-memory communication within a node.

Another distinguishing feature of parallel programming models concerns how threads of execution are managed. For example, in a DM model, it is generally assumed that each thread of execution is a process that always resides on the same processor. In an SMP, there is often a distinction between processes (which maintain an independent data store) and threads (which share a data store), and it is often useful to migrate threads between processors within an SMP. In our survey, we noted that issues of thread management such as thread migration (within an SMP) and thread scheduling have been identified and addressed in specific contexts. However, this work has not addressed the use of threads in the context of scientific computing. For example, prior work has not adequately addressed issues of (1) thread communication across nodes, (2) explicit assignment of thread-processor affinities by users and (3) whether standard scheduling algorithms are appropriate for scientific computing applications.

These considerations lead to the following critiques of the four programming models that we described in our proposal:

- **VN and MVN**: These models should be generally applicable to SMP clusters. However, they are unlikely to take advantage of locality of processes, except through mechanisms in the message-passing interface which recognize when inter-process communication can be performed through a shared-memory operation. The development of numerical libraries which perform basic parallel operations and which take advantage of shared-memory operations may offset this disadvantage, however.

- **SNT**: This model offers the lowest level of interaction with the hardware, and consequently offers the best opportunity for developing highly efficient software. However, it is also a more complicated programming model than VN and MVN. The use of shared-memory programming primitives within for the SMP hardware may hide some of this complexity. Another issue is that this model imposes stronger requirements on the ability of the operating system. Programming models like VN and MVN do not require the management of light-weight threads, since all threads of execution are processes.

- **MNT**: This model offers the additional ability beyond that of SNT for migrating threads across SMP hardware. However, current methods for providing this technique provide the entire machine with a single OS image. This methodology for running the operating system implies a centralization which will make it difficult to scale up to very large machines.

Our investigation into threads packages found that they do not provide the ability for the user to modify the scheduling mechanism except through standard, predefined choices. Thus adapting the scheduling mechanism will require modifications which are not portable to other systems.

Our work on the last two milestones for FY98 is still in progress. First, we are working on a thread-aware implementation of MPI which can take advantage of the extended capabilities that threads provide. Most current vendor implementations of MPI are made to be thread safe, allowing for the message passing in a multithreaded application to function correctly. However, few implementations of MPI exist that have been specifically designed to work with threads. This void can be attributed to several factors. At the time the model implementation of MPI (MPICH [4]) was being developed, a standard API for threads was still under development. Once a standard API for threads was established, the developers of MPICH could not leverage it due to the complexities and peculiarities of each vendor's implementation of threads. Thus, it was left to each vendor to design threads into the existing MPI source. Of the vendors who undertook the task of developing a threaded MPI (and other message passing libraries), most were reluctant to release it to users because the combination of threads and message passing resulted in a significant performance degradation. Essentially, vendors realized that their threads package wasn't designed for a message passing environment

and their message passing package wasn't designed for a threads environment. We are not only working on a message passing environment designed to function in a threaded environment, but also a thread environment designed to function in a message passing environment.

For the last milestone, we have begun to evaluate the requirements for implementing the different programming models. In particular, we have begun to explore how we can leverage the API and constructs of MPI to provide the functionality needed by a hybrid cluster environment. MPI provides objects and operations which encapsulate the elements of a distributed memory system. When dealing with clusters of SMP nodes, MPI will always be utilized for internode communication. Folding shared memory operations and semantics into the MPI environment can provide a common, portable API with which application programmers are already familiar.

For example, a key construct in MPI is the communicator, which represents the individual processes within the application. An MPI process is initially aware of two default communicators: one representing the individual process (MPI_COMM_SELF) and one representing the entire set of processes in the application (MPI_COMM_WORLD). For those processes which may be running on a single SMP node, it may be desirable to have a communicator which represents this set of processes which can share memory, say MPI_COMM_SHARED. The semantics of the operations which occur on such "shared memory" communicators may be different than those which represent disjoint processes. For example, the familiar send and receive operations might move pointers rather than actual data when a shared memory communicator is used. Alternatively, an additional set operations to perform this type of operation might be desired. It may also be desirable to represent a grouping of threads rather than processes. This representation might result in a construct analogous to the MPI group object, but which can be used to identify individual threads. A set of operations to build communicators from this object would then be needed.

The ability to leverage and adapt some of the new operations and capabilities that the MPI-2 standard [7] provides also needs to be examined. For example, MPI-2 provides for the ability to dynamically create processes from within an existing MPI application. Extending the idea of a shared memory communicator, such a spawn operation using this new type of communicator might spawn threads rather than heavyweight processes. This combination of MPI threads and processes may provide a general-purpose methodology for efficiently utilizing all of the processors in an SMP node.

2.2 Proposed Work for FY99

The following milestones are proposed for FY99:

- Complete development of a thread-aware MPI.
- Develop and evaluate application passing interface (API) extensions to MPI for SMP clusters.
- Develop and evaluate a work-proportional scheduler
- Evaluate process and thread schedulers for SMP Linux
- Develop complexity model and prototype primitives for a VN model.
- Demonstrate use of shared-memory phases for dynamic work allocation.

The proposed milestones for FY99 differ significantly from the milestones in the original proposal. In part, these changes reflect the expected difficultly of developing a thread-aware version of MPI. These changes also reflect a focus towards programming models for Cplant™. Unlike the ASCI Red machine, we can readily affect the programming environment in Cplant™ at a very low level (e.g. the operating system). Many of the issues relating to programming SMP clusters relate to the amount of flexibility the operating system and message-passing environment provide, so we expect to make the most progress working with Cplant™.

The first two milestones concern the management of threads and processes. We believe that proportional scheduling will be valuable in the context of scientific computing. It can facilitate gang-scheduling (on shared CPU resources), as well as provide a mechanism for the fair distribution of work between threads within a given application. Our evaluation of schedulers for SMP Linux will include an evaluation of policies for processor-thread affinity.

Finally, the last two milestones focus on the development and evaluation of tools and techniques for specific programming models. First, we will develop efficient primitives and libraries, along with a programming environment, which can support the effective use of SMP clusters for the VN and MVN programming models (with a particular focus on the VN model). Our first phase of this work focuses on the development and prototyping of a complexity model and the primitives. Second, we will develop methods to remedy load imbalance problems by exploiting shared memory parallelism within distributed memory parallelism. This can be done by reassigning processors to processes or by using a general shared-memory phase in which MPI processes take work in a dynamic fashion. This work will help identify the requirements for the SNT and MNT programming models.

3 FY00 Renewal Proposal

3.1 FY99 Accomplishments

For FY99, the initial focus of the project was to address the limitations of current run-time systems in providing an environment flexible enough to support various combinations of threads and message passing. The work planned for FY99 included the development of a thread-aware MPI library, API extensions to MPI, and evaluation of thread schedulers in SMP Linux. Since these goals fall under the area of system software development, the principal investigator was changed in December from Bill Hart to Ron Brightwell and management of the project was moved to the Scalable Computing Systems department, Organization 9223.

Our first milestone, development of a thread-aware MPI implementation, required more effort than was anticipated. As was mentioned in the FY99 proposal, few implementations of MPI exist that have been specifically designed to work in a threaded environment. It was our initial intention to simply modify the public domain version of MPICH to be thread safe. However, it became clear that this approach would be a significant hindrance to the overall goals of this project. In order to fully understand the implications and impacts of combining message passing, shared memory, and threads, an MPI implementation that is fully integrated into a thread environment is required. It is not enough to have an MPI implementation that does coarse-grain locking, or only allows a single thread to perform message passing. Ideally, the MPI library needs to be aware of the threaded environment so that it can take advantage of it. For example, the asynchronous message passing operations in MPI should not waste CPU cycles polling for completion, but rather should allow for descheduling of the calling thread based on completion of the message passing event. This tight integration between threads and MPI is lacking in not only MPICH, but also in most vendor-supplied MPI implementations.

In order to facilitate such an MPI implementation, the underlying message passing layer upon which MPI is built needs to be integrated with threads. We have designed a message passing API, Portals 3.0 [3], to provide this layer. Because Portals are designed to work in a threaded environment, they are ideal not only for supporting MPI, but also for supporting the types of one-sided data movement that may be needed to support a shared-memory style of message passing. In addition to completing the API, we have completed a reference implementation on top of TCP/IP. This implementation is currently thread-safe, and will be fully integrated into a multithreaded environment in Linux before the end of FY99. Portals will be fully functional on Cplant™ on Myrinet [1] networking hardware before the end of CY99.

The design of an MPI implementation for Portals has been completed. This design allows for the MPI library to take advantage of the integration of message passing with threads that Portals provides. Because MPICH has yet to be designed to support threads, we decided to contract with MPI Software Technology, Inc. (MSTI) to provide a thread-aware MPI for Portals. MSTI has a thread-aware MPI implementation for several VIA-based networks using Windows NT. We have provided MSTI with our design of an MPI implementation for Portals, and they have agreed to supply us an thread-aware MPI library for Cplant™ in FY00. They have also agreed to provide us with any extensions to this implementation that we would like to investigate. MSTI may also provide us with the source to the MPI implementation, should they receive ASCI Path Forward funding.

For the milestone that addresses the API extensions to MPI for SMP clusters, we have begun to consider the design of this approach. We hope to have completed an initial study of the constructs that MPI currently provides that can be leveraged to work in clustered SMP environment.

SMP Linux has made great strides during FY99. Any attempt at evaluating thread schedulers in Linux before the current release of version 2.2 would not have been representative of the capabilities of the operating system. We have been waiting for a stable SMP version of Linux to fully test, and plan to upgrade the Cplant™ environment to take advantage of the latest Linux kernels.

We have also made modifications to the run-time environment tools for Cplant™ to allow for investigation of the various programming models outlined in the original proposal. For example, the application launcher (yod) and the compute node process manager (PCT) have been modified so that implementing a virtual node mode (VNM) capability on Cplant™ will be easier.

3.2 Proposed Work for FY00

The proposed milestones for FY00 focus on continuing the development of the Cplant™ machine to support clusters of SMP's. A flexible runtime system that allows an application to choose between combinations of heavyweight processes and threads is highly desirable.

The first three milestones involve extending the Cplant™ runtime environment to support two of the originally proposed programming models. Support for VNM and MVN will allow the user to make a run-time decision about mapping heavyweight processes to processors. The current Cplant™ run-time environment maps a single process to a node, but the components of the run-time system have been redesigned to better accommodate this capability. Support for SNT will be provided by a thread library for Cplant™ that can be used to achieve node-level parallelism.

We will continue to explore how the API and constructs of MPI can be leveraged to provide the functionality needed by a hybrid cluster. We will need to evaluate how our modifications to the semantics of MPI affect the data locality features that MPI provides. Since MPI will always be utilized for internode communication, folding shared memory semantics into the MPI environment may provide a well-known, portable API for programming clusters of SMP's.

With these new capabilities, we hope to demonstrate shared memory parallelism within distributed memory parallelism in addressing load imbalance problems. By reassigning processors to processes or by using a general shared memory phase in which MPI processes accept work in a dynamic fashion, we hope to further identify the requirements for the SNT and MNT programming models.

3.3 FY00 Accomplishments

Due to severe understaffing, the work proposed for FY00 could not be undertaken. The people who were to perform this research were required to dedicate their time to the urgent needs of the Cplant™ project.

References

[1] N. Boden, D. Cohen, R. E. Felderman, A. E. Kulawik, C. L. Seitz, J. N. Seizovic, and W. Su. Myrinet-a gigabit-per-second local-area network. *IEEE Micro*, 15(1):29–36, February 1995.

[2] R. B. Brightwell, L. A. Fisk, D. S. Greenberg, T. B. Hudson, M. J. Levenhagen, A. B. Maccabe, and R. E. Riesen. Massively Parallel Computing Using Commodity Components. *Parallel Computing*, 26(2-3):243–266, February 2000.

[3] R. B. Brightwell, T. B. Hudson, A. B. Maccabe, and R. E. Riesen. The Portals 3.0 Message Passing Interface. Technical Report SAND99-2959, Sandia National Laboratories, December 1999.

[4] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard. *Parallel Computing*, 22(6):789–828, 1996.

[5] A. B. Maccabe, K. S. McCurley, R. E. Riesen, and S. R. Wheat. SUNMOS for the Intel Paragon: A Brief User's Guide. In *Proceedings of the Intel Supercomputer Users' Group. 1994 Annual North America Users' Conference*, pages 245–251, June 1994.

[6] Message Passing Interface Forum. MPI: A Message-Passing Interface standard. *The International Journal of Supercomputer Applications and High Performance Computing*, 8, 1994.

[7] Message Passing Interface Forum. *MPI-2: Extensions to the Message-Passing Interface*, June 1997. http://www.mpi-forum.org.

[8] Sandia National Laboratories. *ASCI Red*, 1996. http://www.sandia.gov/ASCI/TFLOP/ Home_Page.html.

[9] P. L. Shuler, C. Jong, R. E. Riesen, D. van Dresser, A. B. Maccabe, L. A. Fisk, and T. M. Stallcup. The Puma operating system for massively parallel computers. In *Proceedings of the 1995 Intel Supercomputer User's Group Conference*. Intel Supercomputer User's Group, 1995.

Distribution:

1	MS 0188	D. L. Chavez, LDRD Office
1	MS 0310	A. L. Hale, 9220
1	MS 0310	G. S. Heffelfinger, 9209
1	MS 0310	P. Yarrington, 9230
1	MS 0316	J. B. Aidun, 9235
1	MS 0316	S. S. Dosanjh, 9233
1	MS 0318	G. S. Davidson, 9212
1	MS 0318	P. D. Heermann, 9227
1	MS 0321	W. J. Camp, 9200
1	MS 0612	Review and Approval Desk, 9612 For DOE/OSTI
1	MS 0819	E. A. Boucheron, 9231
1	MS 0820	P. F. Chavez, 9232
1	MS 0847	R. W. Leland, 9226
2	MS 0899	Technical Library, 9616
1	MS 1110	D. W. Doerfler, 9224
1	MS 1110	N. D. Pundit, 9223
1	MS 1110	D. E. Womble, 9214
1	MS 9018	Central Technical Files, 8945–1