Cplant™ Runtime System Support for Multi-Processor and Heterogeneous Compute Nodes

Kevin Pedretti[*], Ron Brightwell, and Joshua Williams[†]

June 2002

Abstract

In this paper, we describe additions and modifications to the Computational Plant (Cplant™) system software to support multi-processor compute nodes and to support heterogeneous node types. We describe how these capabilities have been incorporated into our scalable runtime system and how these changes affect the interface seen by end users and application developers. We also discuss several important operating system and networking issues that can directly impact application performance. We present some initial performance metrics that indicate how our current implementation scales when multiple processes are running on a single node.

Keywords: commodity cluster, runtime system, multi-processor, heterogeneous computing

1 Introduction

Clusters of commodity PC hardware connected by gigabit network hardware running open source operating systems have become one of the most cost effective platforms for parallel scientific computing. Clusters consisting of thousands of processors have been deployed at several sites, and many of these systems have been ranked among the most powerful computing systems in the world.

Most of these clusters utilize multi-processor compute nodes, which have the potential to deliver more cost-effective performance than clusters based on single-processor compute nodes. The ability to deploy a scalable runtime system that supports multi-processor compute nodes is an important requirement for many sites. Typical multi-processor compute node contain between two and eight processors that share access to memory, disks, and network interfaces. Because of the extra processing power and economy of scale, multi-processor nodes typically have a lower price-performance ratio than single-processor nodes.

Unlike vendor proprietary parallel systems of the past, commodity clusters are usually composed of small building blocks of components, typically individual nodes or racks of nodes. This characteristic allows cluster systems to be expanded more easily and more often. Unfortunately, such expansion usually leads to a mixture of heterogeneous components. Even a single large order of PC's has been known to arrive with small variations in internal hardware and software components, such as an upgraded version of the BIOS. For sites that wish to deploy large-scale commodity cluster systems, heterogeneity is inevitable.

Sandia National Laboratories has developed the Computational Plant (Cplant™) system software to enable clusters of commodity PC's to scale to the order of 10,000 nodes. As of April 2002, the largest cluster running Cplant™ is comprised of 1,792 single-processor compute nodes operating in a near production quality state with more than one hundred users. Flexibility and portability

[*]K. Pedretti and R. Brightwell are with the Scalable Computing Systems Department, Sandia National Laboratories, PO Box 5800, Albuquerque, NM, 87185-1110, (505)845-7397, (505)845-7442 FAX, {ktpedre,rbbrigh}@sandia.gov. 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.

[†]J. Williams is with Unlimited Scale, Inc., PO Box 22409, Eagan, MN, 55122-0409, (651)554-0171, (651)554-0176 FAX, jw@unlimitedscale.com

are two important features of Cplant™. It has been ported to various processor architectures (Alpha, x86, IA-64) and can use various interconnects (Myrinet [1], Ethernet, and soon Quadrics [2]). The Cplant™system software is available as open source.

This paper presents additions to the Cplant™ runtime system that enable the use of multi-processor compute nodes. Additionally, new features that add the ability to use heterogeneous nodes of the same architecture will be described (e.g., all Alpha nodes with differing amounts of memory). The following section describes the architecture of a Cplant™ cluster. Section 3 describes the evolution of the Cplant™ runtime system and discusses some of the factors that inhibited the use of multi-processor systems and heterogeneous systems. In Section 4 we present our approach to add multi-processor and heterogeneous node support to Cplant™. In Section 5, we briefly present initial results obtained with the current implementation. We summarize in Section 6.

2 Cplant™ Architecture

In the following sections, we limit the discussion of the Cplant™ architecture to the salient features relevant to the design for multi-processor and heterogeneous node support. A more detailed description of the architecture can be found in [3, 4, 5].

Cplant™employs the partition model of resource provision [6] in which nodes in the cluster assume specialized roles. There are three types of nodes relevant to this discussion. Administration nodes are dedicated solely to management tasks that act upon all of the nodes in the system. Service nodes provide a full set of features and are where users launch parallel jobs. Compute nodes are optimized for running parallel processes and typically have a stripped-down or lightweight operating system. Current Cplant™ systems use a minimal Linux kernel, but the use of a Sandia-developed lightweight kernel is being actively pursued.

A Cplant™ cluster is typically composed of *scalable units* (SU's). Each SU contains one administration node to manage all of the other nodes. The rest of the nodes can be configured as compute nodes or service nodes. Several SU's can be combined to form a single Cplant™ system. For example, a cluster could be composed of four SU's each with 32 service nodes and 32 SU's each with 32 compute nodes. With single-processor compute nodes, such a system could run applications on up to 1024 processors. The administration nodes are typically connected in a hierarchical network so that the cluster can be administrated as a whole from a single node at the root of the tree.

The distinction of roles and the concept of the SU are key components that enable Cplant™ systems to scale to thousands of nodes. However, smaller Cplant™ clusters may be more ad-hoc and contain nodes that perform multiple roles. For example, Figure 1 depicts a four node Cplant™ where node 0 performs all three roles, nodes 1 and 3 operate as service and compute nodes, and node 2 functions solely as a compute node.

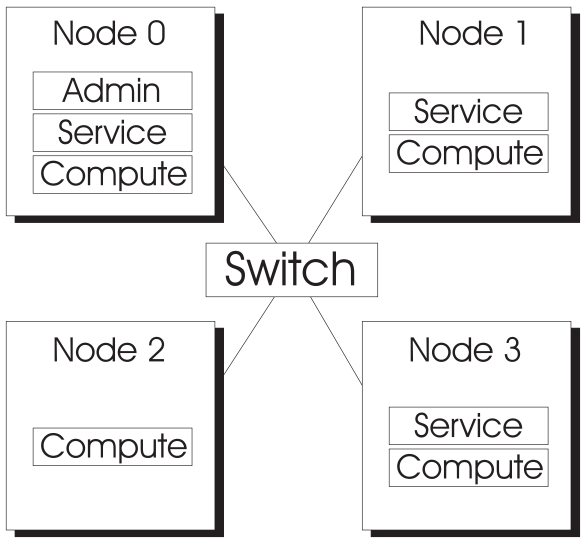

Figure 1: Node Roles in a Small Cplant™

Users submit jobs to a Cplant™ system from a service node via the `yod` command. Command line options are used to specify information such as how many compute nodes the application requires, which nodes to use, and whether or not to start a debugger. `yod` contacts the resource allocator, `bebopd`, to request nodes for the application that it is attempting to start. If `bebopd` is able to satisfy the request, it returns a list of compute nodes to `yod`. `yod` then contacts the `Process Control Thread` (PCT) daemons that `bebopd` has allocated and directs them to form a spanning tree. PCT daemons run on compute nodes and are responsible for loading, executing, and managing (e.g., delivering signals) application processes. In a Cplant™with single-processor nodes, one PCT runs on each compute node. The spanning tree of allocated `PCT's` is used by `yod` to efficiently distribute the application's environment and executable (or executables in the case of a multi-executable application). Once all of the necessary data is distributed, `yod` directs the `PCT's` to start the application. Since the executable image is distributed to the compute nodes by the runtime system, no shared filesystem between service nodes and compute nodes is required. The Cplant™runtime system has demonstrated launching a parallel application on more than 1000 nodes in a matter of seconds [4]. Once an application is running, the `pingd` command can be used to query the system to obtain the status of a job.

Figure 2 illustrates the interaction of the Cplant™runtime components described above. In this example, the `bebopd` node allocator runs on node 0. The `yod` programs running on nodes 0, 1, and 3 communicate with the `bebopd` to request compute nodes (i.e., nodes running a `PCT`). Here, the `bebopd` allocated the `yod` on node 0 the `PCT` on node 0. Similarly, the `yod` on node 1 was allocated the `PCT's` on nodes 2 and 3. For this example, node 0 is the admin node, nodes 0, 1, and 3 are service nodes, and all four nodes are compute nodes.

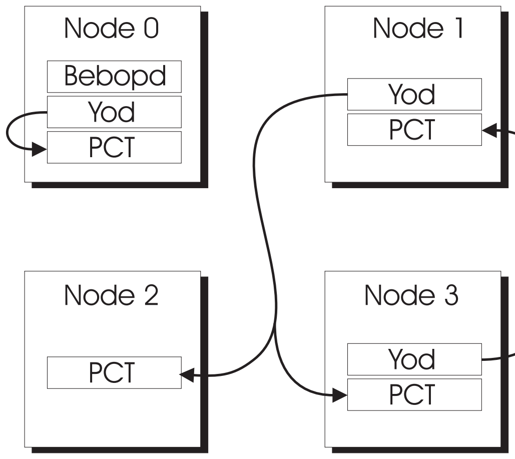

Figure 2: Runtime Daemons in a Small Cplant™

Each processes in a job is configured to have standard input and standard output redirected to the `yod` process that created the job. This allows users to provide input to and observe output from all of the processes in a job from a single process. For high-performance parallel I/O, processes can perform file I/O operations with a Sandia-developed filesystem called ENFS, which provides a shared filesystem for parallel independent I/O.

All communication between nodes is performed via the Portals 3.0 data movement layer [7]. Portals provides an interface and semantics sufficient to allow operating system (OS) bypass and application offload, both of which increase the potential for overlap of computation and communication. It also is designed to be highly scalable. Each of the runtime components mentioned in the previous section use a library layered on top of Portals for communication. Parallel applications typically use an MPI [8] implementation based on MPICH [9] that has been ported to use Portals as the underlying communication mechanism.

Our production Cplant™ clusters currently use a kernel-based implementation of Portals that works with any network device that Linux supports. This is accomplished via two kernel modules, the Portals module and the RTS/CTS module. The RTS/CTS module uses any Linux networking device that provides raw packet delivery. RTS/CTS provides packetization, network reliability, and flow control services. This module works in conjunction with a separate kernel module that implements Portals' semantics. While this particular implementation does not allow for OS-bypass, we currently have several implementations in development that do.

3 Limitations

In this section we discuss the current limitations of the Cplant™ runtime system in supporting multi-processor nodes and heterogeneous node types. While our original intention was not to restrict the Cplant™ runtime system to only support single-processor nodes, many factors contributed to this limitation as the system evolved.

3.1 Multi-Processor Compute Nodes

The Cplant™ runtime system evolved from the runtime system developed on previous large-scale distributed-memory parallel computing platforms, such as the Intel Paragon and Intel ASCI/Red machines, which used a Sandia-developed lightweight kernel, called Puma [10], for compute nodes. Even though both of these systems were composed of dual-processor compute nodes, Puma was not originally designed to allow two application processes to run concurrently on both processors on a node. Puma originally supported three modes of operation [11]. In the first mode, the application processes and the kernel shared a single processor while the second processor went unused. In the second mode, the kernel runs on the first processor and the application processes run on the second. In the third mode, the kernel and application processes run on the first processor and application processes can start co-routines on the second processor. Because we tried to leverage this software as much as possible when the Cplant™ project was begun, the limitations of the lightweight kernel's usage model with respect to multi-processor nodes were continued[1].

In early 1997 when the Cplant™ project began, single- and dual-processor PC systems were evaluated to determine which system provided the best performance for Sandia's important applications. Since our applications are typically memory bandwidth intensive, single-processor Alpha-based systems outperformed both single-processor and dual-processor x86-based systems. The memory subsystem of the Intel-based PC's were inferior to the Alpha systems. That trend continues to the present. As dual-processor Alpha systems became available, there were no apparent cost advantages to acquiring these systems for Cplant™. The cost savings of dual-processor Alpha systems were not realized when considering other factors, such as their decreased density. For these reasons, all production Cplant™ clusters have been composed of single-processor Alpha-based systems, and the motivation to develop the runtime system to support dual-processor systems was low.

Another contributing factor was Linux's lack of support for multi-processor systems in 2.0 and 2.2 versions. It has only been recently that the Linux 2.4 kernel has appropriately addressed issues for multi-processor nodes. Early attempts at using dual-processor systems with pre-2.4 kernels were largely unsuccessful and the cost/performance savings of dual-processor systems were essentially unrecognized due to the instability of Linux.

[1] A fourth processor mode that supports running the kernel and an application process on the first processor and another complete application process on the second processor was developed and deployed in 1999. This new mode, called virtual node mode, allows a single compute node to be seen as two separate single-processor compute nodes.

3.2 Heterogeneous Compute Nodes

Even though the original Cplant™concept involved a strategy for growing the cluster with new hardware while pruning off obsolete hardware, we found this very difficult to do in practice for a production computing platform. Experience with heterogeneous compute nodes on previous systems did not encourage such systems. For example, our large Intel Paragon system was originally delivered with computes nodes that contained 16 MB of main memory. The system was subsequently upgraded, and compute nodes with 32 MB of main memory were added. Even though this information was exposed to applications through a non-standard programming library interface, no applications were adapted to the differences in memory size. Application developers simply viewed the node attributes by their "lowest common denominator".

The advent of cluster computing has brought the need for supporting heterogeneous systems to the forefront. There are now several projects that are addressing the ability for parallel applications to make intelligent decisions regarding load-balancing on heterogeneous computing systems. One such project at Sandia is Zoltan [12].

In addition to the factors listed above, there were many others that steered development of Cplant™ toward single-processor homogeneous systems. However, we believe that support for multi-processor nodes and heterogeneous systems is key to wide-spread use of the technology we have developed.

4 Approach

In this section we discuss our approach to supporting multi-processor nodes and heterogeneous systems within the Cplant™runtime system. We present some design alternatives and provide an in-depth description of our chosen approach.

4.1 Multi-Processor Support

Two approaches for adding multi-processor support were identified. First, either the `PCT` could be extended to manage multiple user processes per node or the runtime system could be modified to support multiple `PCT's` on each compute node. These approaches will be termed *OPPN* (One PCT Per Node) and *OPPC* (One PCT Per CPU) and are shown in Figure 3.

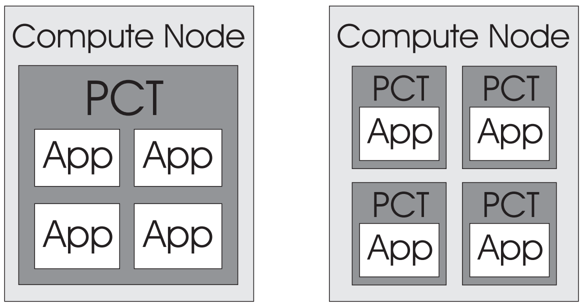

Figure 3: OPPN vs. OPPC

The OPPN approach has the advantage that it uses fewer resources per node. It is more difficult to implement than OPPC because it requires significant modifications to the current `PCT`, since it was developed with the assumption that it would only manage a single application process. OPPC consumes more resources, but requires relatively minor changes to the runtime system. Both approaches require that the Linux kernel modules that implement the Cplant™ name space, Portals API, and low-level RTS/CTS protocol be made re-entrant.

Sandia chose to implement the OPPN approach because it is more closely aligned approach of the lightweight kernel. The OPPN approach is also more amenable to Sandia's plans of running a lightweight kernel on compute nodes. The design of the lightweight kernel does not allow for sufficient functionality to partition resources (e.g. memory, processor time) between multiple `PCT's` efficiently. In fact, in the lightweight kernel, the `PCT` is responsible for allocating *all* of the compute node resources, so multiple `PCT's` would be redundant.

In a non-multi-processor Cplant™, users specify to `yod` the number of compute nodes needed with the `-sz` argument. For example, a command to launch an application `app` on 32 nodes would look like '`yod -sz 32 app`'. In an multi-processor cluster, the term "*node*" becomes less clear. There is no longer a one-to-one correspondence between the number of nodes allocated and the number of nodes in the job. With this limited job size specification, it is up to the `bebopd` to decide which `PCT` compute nodes get allocated and how many processes are to be run by each `PCT` in the case of OPPN.

To support multi-processor compute nodes, the size (`-sz`) argument to `yod` has been extended to optionally specify the number of processes per node (ppn) or total number of processes requested (procs). The tuple $\{nodes : ppn : procs\}$ specifies how to load the application. In a fully specified size argument (e.g., '`yod -sz 32:4:128 app`'), $nodes * ppn$ must equal *procs*. Any valid combination of arguments is allowable. For example, if the user wants an application to contain 32 processes but does not care how many physical nodes are allocated, they can indicate this by '`yod -sz ::32 app`'. Other users may want to ensure that each process in the application gets exclusive access to the resources on a physical node (e.g., memory bus, network interface) and specify the number of nodes and the number of processes to run on each node (e.g., '`yod -sz 32:1 app`' which is equivalent to '`yod -sz 32 app`' for backward compatibility).

To support multi-processor compute node, the runtime system restricts compute nodes to running processes from a single application at any given time. Sharing processors on a compute node between multiple parallel jobs violates our space-sharing philosophy of managing a system, and would require significant changes to components outside the runtime system, such as the batch scheduler and job logging system.

4.2 Heterogeneous Support

The need has arisen at Sandia to support a cluster with compute nodes that have differing amounts of memory. The current approach is to split a physical cluster that contains heterogeneous nodes into *virtual machines* where each virtual machine contains a set of homogeneously configured nodes. Each virtual machine looks like an independent machine, and it is not possible to run a job that spans two virtual machines.

To avoid the need to artificially partition heterogeneous nodes, the concept of node attributes has been introduced.

Abstract attributes and attribute values can be defined by an administrator and assigned to compute nodes. Figure 4 gives examples of two new Cplant™ configuration files, `cplant-attribs` and `cplant-nodespec`, that accomplish this. The `cplant-attrib` file defines the attributes recognized by the system. Each attribute and value text string are purely abstract and are converted to integer ids for use internally by the Cplant™ runtime system. In the figure, three attributes are defined: CPU, MEM, and NIC. The second column of each attribute definition specifies the valid operations for the attribute. The operations are used by user's to specify resource requirements when submitting jobs.

Following the operations, is a list of values for the attributes. The `cplant-nodespec` file is used to assign attribute-value pairs to compute nodes. The first column specifies the node id. The second column specifies the maximum number of processes to run on the corresponding compute node (node width). Following this is a list of attribute value pairs. Every attribute must be defined for each node listed in the `cplant-nodespec` file. Nodes not explicitly listed will obtain a default node width of one and the first value defined for each attribute in the `cplant-attrib` file.

```
cplant-attribs:

CPU  =,>=,<=   500MHZ 1GHZ 2GHZ
MEM  =,>=,<=   256MB 512MB 2GB
NIC  =         FAST   SLOW

cplant-nodespec:

0   4  CPU=500MHZ, MEM=512MB, NIC=SLOW
1   4  CPU=1GHZ,   MEM=512MB, NIC=SLOW
2   4  CPU=2GHZ,   MEM=1GB,   NIC=FAST
10  2  CPU=2GHZ,   MEM=2GB,   NIC=FAST
```

Figure 4: Heterogeneous Node Configuration Files

A new argument has been added to the `yod` command line to allow users to specify the type of nodes on which to run their application. It is instructive to use an example to illustrate. Consider a Cplant™ with the attributes defined by the configuration files in Figure 4. If a user wanted to launch an application `app` on two 512MB, slow network nodes, and have 2 processes per node, they could specify this by:

```
yod -sz 2:2 -na "MEM=512MB, NI=SLOW" app
```

In this case, the only nodes meeting this criteria are nodes 0 and 1. The `Bebopd` would thus be required to assign these two nodes. The ">=" and "<=" operators provide more flexibility to users when specifying requirements. If the requirement is compute nodes with *at least* 512 megabytes instead of *exactly* 512 megabytes and the speed of NIC isn't important, the following could be specified:

```
yod -sz 2:2 -na "MEM>=512MB" app
```

In this case, all four nodes are candidates and the `bebopd` has considerably more flexibility. With this scheme, it is in the user's best interest to specify job requirements as loosely as possible. Very explicit resource requests will likely take longer to fulfill and may significantly degrade the performance of the runtime system.

5 Initial Implementation and Results

This section describes the current state and initial results of Sandia's multi-processor implementation. As of June 2002, the implementation appears to be stable based on limited testing on a 4-node cluster. All of the modifications necessary to the runtime programs (i.e., `bebopd, yod, PCT, pingd`) have been completed and verified to work properly when running multiple processes per node. The multi-processor modifications to the Cplant™ modules were initially problematic. We believe we have identified and resolved the race conditions that were causing these problems. We hope to soon begin more extensive testing on a 128-node cluster composed of dual Pentium IV nodes.

The additions described in this paper have been developed and tested on an Itanium [13] cluster at the University of New Mexico. The cluster consists of four Hewlett Packard rx4610 servers, each with four 733 MHZ processors and four gigabytes of memory. It is not the purpose of this paper to evaluate the performance of multiprocessor systems in a general manner. However, we have seen some interesting results.

We've implemented intra-node messages via shared memory. Initial benchmarks show that MPI latency for zero-length intra-node messages over 100Mb/s Ethernet is 30 μsec and inter-node messages is 101 μsec. With the shared memory message path turned off (i.e., messages go through the networking system), latency rises to 72 μsec for intra-node messages. For applications where only latency is critical, this shows a clear benefit for taking the shared memory approach. Once our Myrinet drivers are ported to IA-64, we expect to get much better latency results.

Locking has been implemented so that only one process is allowed to use the Cplant™ modules at any given time. A Linux *spin-lock* is obtained at every entry point into the modules and released when the critical section is over. When entering the Cplant™ modules from a user-context, interrupts on the entering CPU are disabled. This avoids deadlock with the RTS/CTS receive interrupt handler, which acquires the global spin-lock asynchronously. While this approach was identified as the most rapid way to get the modules working on multi-processor machines, finer-grained locking would provide opportunities for pipelining send processing and this greater performance. Figure 5 compares the performance the NAS NPB [14] LU benchmark running under Cplant™ and running under a native version of MPICH 1.2.3. Single process performance between the two is similar. For the case where four processes run on the same node, native MPICH is considerably faster than Cplant™. The gap narrows when four processes are run with two processes per node. This seems to indicate that there is contention among the processes for access to the Cplant™ networking modules caused by the global spin-lock. Finer-grained locking, where multiple processes can be allowed to use the Cplant™ networking layer simultaneously, should alleviate this problem.

	Cplant Total	Native MPICH Total	Cplant Per Proc	Native MPICH Per Proc
1 Process	63.4	65.6	N/A	N/A
4 Procs, Same Node	195.11	244.9	48.8	61.2
4 Procs, Two per Node	215	230.3	53.75	57.6

Figure 5: Cplant™ vs. Native MPICH for LU NBP Benchmark (in Mop/s)

In the future, we will be comparing the performance of the OPPN approach to the OPPC approach. Metrics such as job load time, compute node memory usage, and compute node processor overhead will be examined.

6 Conclusion

In this paper, we have briefly described the Cplant™ system software and additions to it for multi-processor and heterogeneous compute node support. Changes to the user interface for submitting jobs has been described along with examples. Preliminary results show good latency for intra-node messages. However, we have identified possible contention for the Cplant™ networking layer when there are are multiple processes on the same node communicating due to our simple locking implementation. This should be remedied when finer-grained locking is in place.

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*, vol. 15, no. 1, pp. 29–36, February 1995.

[2] F. Petrini, W. Feng, A. Hoisie, S. Coll, and E. Frachtenberg, "The Quadrics Network: High-Performance Clustering Technology," *IEEE Micro*, vol. 22, no. 1, pp. 46–57, January 2002.

[3] 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*, vol. 26, no. 2-3, pp. 243–266, February 2000.

[4] R. B. Brightwell and L. A. Fisk, "Scalable Parallel Application Launch on Cplant™," in *Proceedings of the SC2001 Conference on High Performance Networking and Computing*, November 2001.

[5] R. B. Brightwell and S. J. Plimpton, "Scalability and Performance of Two Large Linux Clusters," *Journal of Parallel and Distributed Computing - Special Issue on Cluster and Network-based Computing*, vol. 61, no. 11, pp. 1546–1569, November 2001.

[6] D. S. Greenberg, R. B. Brightwell, L. A. Fisk, A. B. Maccabe, and R. E. Riesen, "A System Software Architecture for High-End Computing," in *Proceedings of the SC'97 Conference on High-Performance Networking and Computing*, November 1997.

[7] R. Brightwell, W. Lawry, A. B. Maccabe, and R. Riesen, "Portals 3.0: Protocol Building Blocks for Low Overhead Communication," in *Proceedings of the 2002 Workshop on Communication Architecture for Clusters*, April 2002.

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

[9] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, "A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard," *Parallel Computing*, vol. 22, no. 6, pp. 789–828, September 1996.

[10] 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.

[11] A. B. Maccabe, R. E. Riesen, and D. W. van Dresser, "Dynamic Processor Modes in Puma," *Bulletin of the Technical Committee on Operating Systems and Application Environments (TCOS)*, vol. 8, no. 2, pp. 4–12, 1996.

[12] K. Devine, E. Boman, R. Humphrey, B. Hendrickson, and C. Vaughan, "Zoltan Data Management Service for Parallel Dynamic Applications," *Computing in Science and Engineering*, vol. 4, no. 2, March/April 2002.

[13] B. Greer, J. Harrison, G. Henry, W. Li, and P. Tang, "Scientific Computing on the Itanium Processor," in *Proceedings of the SC2001 Conference on High Performance Networking and Computing*, November 2001.

[14] D. H. Bailey et al., "The NAS Parallel Benchmarks," *International Journal of Supercomputer Applications*, vol. 5, no. 3, pp. 63–73, 1991.