

PVM Communication Performance in a Switched FDDI Heterogeneous Distributed Computing Environment

Michael J. Lewis

Distributed Computing Department
Sandia National Labs
Livermore, CA 94550

Raymond E. Cline, Jr.

Distributed Computing Department
Sandia National Labs
Livermore, CA 94550

Abstract

The Parallel Virtual Machine (PVM) message passing system developed at Oak Ridge National Laboratories has gained widespread acceptance and usage in the parallel programming community. This paper describes the results of performance tests of PVM in a switched FDDI heterogeneous distributed computing environment. We aim to provide insight into how parallel programs, particularly those employing PVM for intertask communication, will perform in distributed systems of the future.^{1 2}

1 Introduction

The past several years have seen an intense effort to harness the power of networked workstations as an alternative to a single costly high-speed computer. However, the efficacy of parallel programming in a distributed system is limited by several factors. These include 1) the nature of the problem being attacked, 2) the design of the application program, 3) the speeds of the various components of the network, and 4) the communication between nodes in the system. This work investigates the latter two. Our results will provide insight into the communication performance of parallel applications running on distributed systems which reflect today's latest network and workstation technology.

For distributed systems to successfully compete with high performance computers, we believe that memory to memory throughput must exceed 500 megabits per second (Mbps) and communication latencies must be kept below 100 microseconds. We base this on our analysis of numerous representative distributed applications including regular 3D finite element and 3D molecular dynamics using a space cell decomposition. This communication performance is required to make local computation time the limiting factor even in the most communication intensive portions of the applications. More details can be found in [4]. To meet these goals, performance enhancements

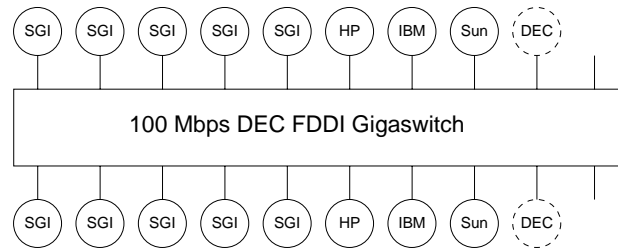


Figure 1: HEAT network configuration

must be made in the following areas: 1) physical networks, 2) communication protocols, and 3) message passing libraries. For each of these, we plan to characterize existing technology and investigate alternate directions. This work describes preliminary results as we begin this process. In particular, we present the results of real experiments in a state of the art distributed computing environment.

1.1 The HEAT network

Our test environment, called HEAT (Heterogeneous Environment and Testbed), is a network of 50 workstations consisting of 10 of each of the following: IBM/RS6000's, HP-9000 PA-RISC's, SGI IRIS Indigos, DEC Alphas, and Sun SPARCstation 10's. These workstations are networked via a 100 Mbps DEC FDDI Gigaswitch and several DEC Concentrator 500's. Preliminary performance testing shows that two Indigo workstations can use TCP to push up to 93 Mbps through the switch[1]. We used a subset of HEAT for the performance tests in this paper. This subset (depicted in figure 1) contained 16 workstations: 10 SGI's, 2 IBM's, 2 HP's, and 2 Suns. (At the time the tests were run, the network was being upgraded to include the DEC's.)

1.2 PVM

Performance measurements were taken using a simulation program written with the Parallel Virtual Machine[2][3] (PVM) message passing library. PVM is designed to facilitate the design and implementation of applications which wish to exploit a heterogeneous set of networked machines. The PVM library available to the C or Fortran application programmer includes

¹This research was supported by the United States Department of Energy under contract #DE-AC04-76DP00789.

²Appeared in *Proceedings of the IEEE Workshop on Advances in Parallel and Distributed Systems*, October 1993

routines to spawn and kill processes, pack and unpack message buffers, send and receive messages, and synchronize processes through group communication. Users can configure their own pool of processors and may either assign machines to tasks or allow PVM to do so. Task to task communication is realized in one of two ways. By default, messages are sent through a local TCP socket to the PVM daemon process (`pvmd`) on the node where the message originates. From there, they are forwarded to the remote `pvmd` via a UDP network socket, and then passed on to the remote task through another local TCP socket. As an alternative, the user can cause the `pvmds` to be bypassed by having PVM set up direct TCP connections between tasks.

2 Performance tests

This work is not an attempt to accurately model real distributed programs. Instead, we have identified the major characteristics of a PVM application which affect its *communication* performance. We model a system of tasks communicating with extremely fine grained parallelism; no time is spent on computation at the nodes. Our simulation program, `pvmtime`, spawns a system of processes which interact with one another based on a set of program parameters. Those which were varied for this paper are described below.

- **test type:** `pvmtime` can run either of two types of tests, *Ping Pong* or *Blast*.
 - The *Ping Pong* test starts one or more master tasks which receive and echo packets from slave tasks; the slaves wait for the echo before sending subsequent packets.
 - The *Blast* test starts masters which receive the slaves' packets without echoing them.
- **PVM routing style:** Either of PVM's two routing styles (TCP direct or `pvmd`) may be selected.
- **system configuration:** The number and location of the tasks (masters and slaves) are set by a configuration file which is read at runtime.
- **message packing:** `pvmtime` allows 16 different "pack levels" characterized by whether masters and slaves pack and unpack messages. We report the results for two pack levels: 1) no packing or unpacking of messages is done, and 2) each message is packed by its sender and unpacked by its receiver.
- **packing style:** PVM supports two different styles of packing messages. `PvmDataRaw` can be used to send unencoded data between common machine types. `PvmDataDefault` will cause PVM to use XDR encoding to allow communication in a heterogeneous system. (`PvmDataInPlace` was not implemented at the time these tests were run.) Both packing styles were tested and are compared in section 3.2.
- **network MTU size:** PVM fragments large messages into smaller ones before transmitting the

data on the network. We altered the PVM source code and tested performance for 8K and 12K fragment sizes in addition to the default 4K fragments.

3 Results

For the tests reported here, each slave sent 1024 messages to its master. The message size ranged from 4 bytes to 16K bytes and varied in intervals of 1K bytes. In the *Blast* test, the slaves finish sending all the messages before they are delivered to the master at the destination. Thus, throughput would appear higher in measurements taken at the slaves. To represent the true end-to-end performance, we report the throughput measured at the master process, including in the calculation both incoming and outgoing messages. In the case of multiple masters, the average of their measured throughputs is reported.

For the results included below, unless otherwise specified, the following are assumed: a single master/slave pair, no two tasks on the same machine, a network MTU of 4K bytes (PVM's default), no packing or unpacking of messages, and `PvmDataDefault` (XDR encoded) packing when it is done. Note that each of these is altered in at least one set of graphs. The test type and routing style is explicitly noted in the caption of each graph. The y-axis scale is roughly the same for all graphs, allowing direct comparisons between curves in separate figures.

3.1 SGI tests

The results presented in this section refer to tests which sent data between SGI IRIS Indigo machines directly connected to the FDDI switch.

3.1.1 Routing style

We tested the relative performance of PVM's two different routing styles by running *Blast* and *Ping Pong* for both `pvmd` and TCP direct routing. During the tests it was noted that the `pvmd` daemon often died when attempting to forward a large number of sizeable messages. We did not investigate this situation in detail, but it seems to be a flow control problem within PVM. Therefore, throughout the paper, we omit all results of the *Blast* test through `pvmds`. We investigated the scalability of each routing style by replicating tests using one and four slaves. The results of these tests are shown in figures 2 and 3.

In figure 2, the two TCP curves have a slightly positive slope whereas the `pvmd` curves are relatively flat. A smooth curve fit to **TCP 4 Slaves** would seem to be three to four times higher than one fit to **TCP 1 Slave**. Going from one to four slaves with `pvmd` routing only results in a 25% to 40% increase in throughput. These observations indicate that applications using TCP routing will scale with network usage better than those which employ `pvmd` routing.

The *Blast* test (figure 3) achieves significantly higher throughput than *Ping Pong*. Similar to figure 2, throughput with TCP routing and four slaves is nearly four times greater than with a single slave. However, the **TCP 1 Slave** curve has a negative slope

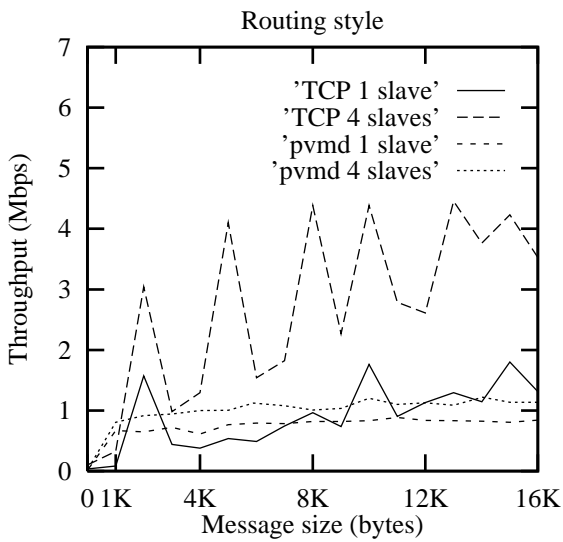


Figure 2: *Ping Pong* test, TCP and pvmd routing.

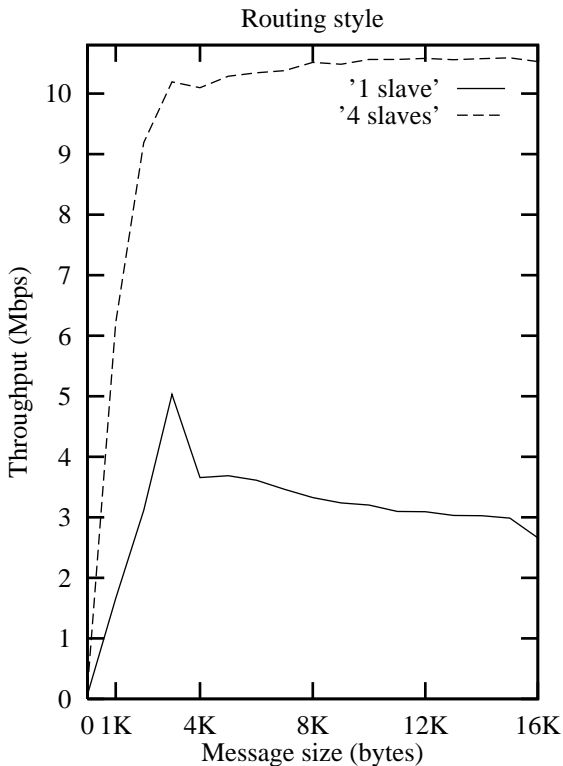


Figure 3: *Blast* test, TCP routing.

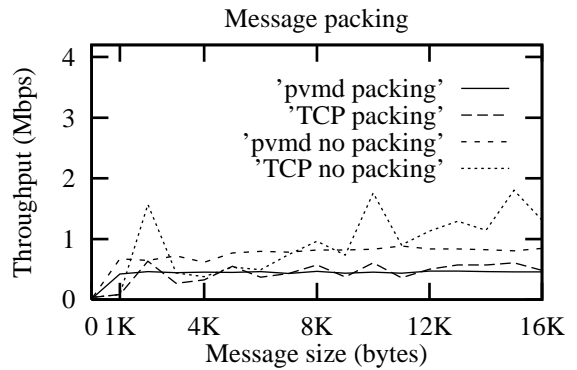


Figure 4: *Ping Pong* test, TCP and pvmd routing.

which appears to contradict the notion that applications using TCP direct routing will scale well with network usage. As will be seen in later tests, this negatively sloped curve is particular to a single master/slave pair running on SGI machines, and is possibly attributable to SGI's unique TCP/IP implementation.

The appearance of multiple spikes in the TCP curves in figure 2 is something which is repeated in several subsequent graphs. We attribute these to the machines' internal memory management buffers and how their sizes relate to `pvmttime`'s message size. The spikes are more common in the results of the *Ping Pong* test, perhaps due to the fact that memory management buffers must be shared between incoming and outgoing messages.

3.1.2 Message packing

To test the effects of the PVM message packing routines, we ran the two test types for a single master/slave pair. Figures 4 and 5 compare the results of packing and unpacking all messages as opposed to doing neither. All packing is done on integer data and uses XDR encoding.

Figures 4 and 5 indicate that message packing causes an average performance degradation of nearly 50% with pvmd routing, and that throughput of the TCP *Ping Pong* and *Blast* tests are limited to about 0.6 Mbps and 1.0 Mbps respectively. More observations about the effects of message packing are included in section 3.2.

3.1.3 Cross traffic

We also tested for performance degradation due to cross traffic. This was done by starting multiple masters, each with its own set of slaves. Figures 6, 7, and 8 depict the results of these tests. In the notation used to label the graphs, 4.4-4.4 means that four masters on four different machines communicated with a total of four slaves (one per master) on four other machines. 4.1-4.1 indicates that four masters shared a

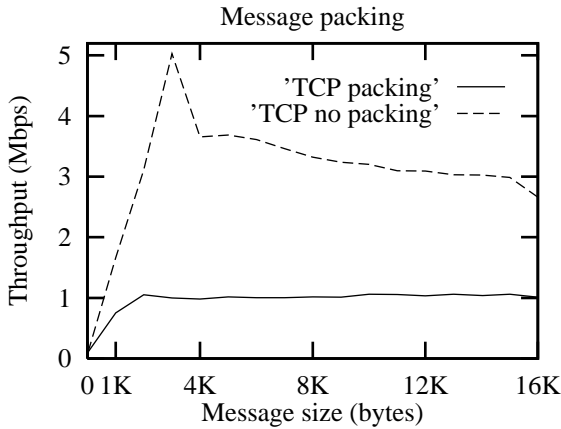


Figure 5: *Blast* test, TCP routing.

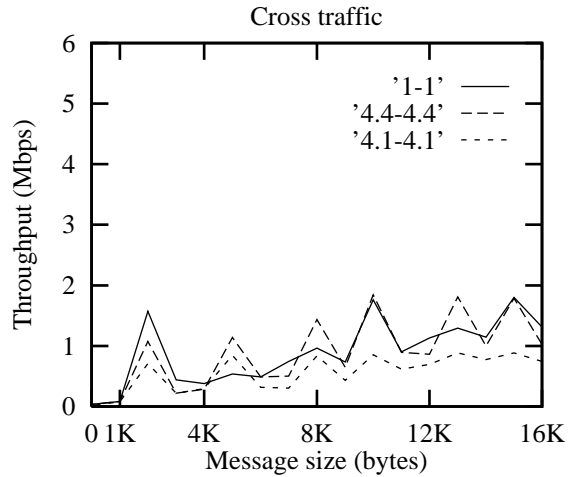


Figure 7: *Ping Pong* test, TCP routing.

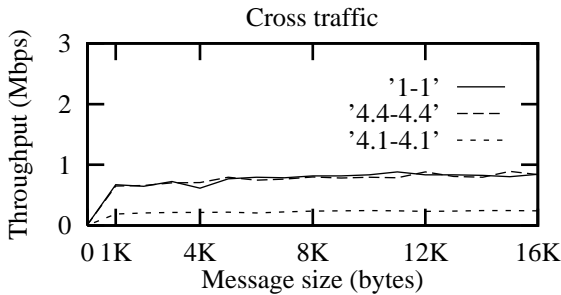


Figure 6: *Ping Pong* test, *pvmd* routing.

single machine and communicated with a total of four slaves sharing a different machine. The single master/slave pair (no cross traffic) is included for comparison purposes. We report the average throughput of all masters in each particular test.

The 1-1 and 4.4-4.4 curves in figure 6 are quite similar; so too (although less noticeably so because of the spikes) are the 1-1 and 4.4-4.4 curves of figure 7. This indicates that performance of the *Ping Pong* test is unaffected by external traffic in the system. Each 4.1-4.1 curve is lower than the others in the same graph, with the decrease in *pvmd* routing being much more pronounced. This shows that the *pvmd* (and to a lesser extent the CPU) becomes a bottleneck in the *Ping Pong* test when multiple tasks reside on the same machine.

The gap between the 1-1 and 4.4-4.4 curves in figure 8 indicates that the network itself can limit the performance of the *Blast* test even at throughputs significantly lower than the network's capability. The gap is larger for small messages and narrows with increased message size. This seems attributable to the fact that the Gigaswitch is asked to set up and tear down virtual connections at a faster rate when small

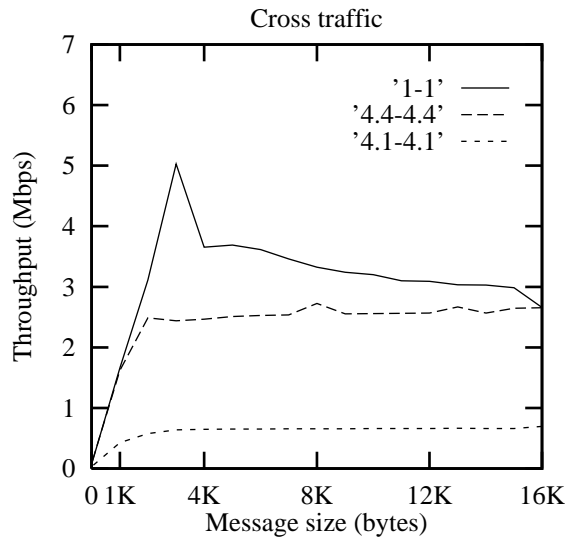


Figure 8: *Blast* test, TCP routing.

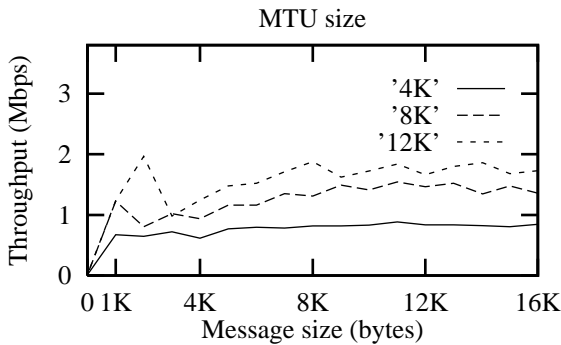


Figure 9: *Ping Pong* test, *pvmd* routing.

messages are being transmitted.

3.1.4 MTU size

PVM source code was altered to change the size of the packets into which large messages are fragmented. The legends of figures 9 and 10 indicate the three different MTU sizes we tested. We report the results of the *Ping Pong* test through *pvmds*, and the *Blast* test using TCP routing.

Figures 9 and 10 indicate that throughput increases with MTU size for both routing styles. The appearance of spikes in figure 10 supports the notion that they are caused by the relationship between message size and the memory management system's buffer sizes; smaller fragments (4K) are more likely to completely fit in the machines' buffers at the various memory management levels, whereas 8K and 12K chunks may need to be split across multiple buffers.

3.2 HP, IBM, and Sun tests

To test PVM's performance on the various machines, we repeated the *Blast* test with TCP routing and *Ping Pong* with *pvmd* routing for each of the four workstation types in our network. Tests were run for a single master/slave pair with each of the three packing styles described in section 2. A comparison between the performance of PVM on the different architectures is displayed in figures 11 through 16.

These results indicate that, among the four workstation types which were tested, PVM performs best on the HP's. Five of the six tests ran slowest on Sun machines. The IBM's and SGI's performed about equally on our tests.

There is a noticeable performance degradation incurred as a result of XDR encoding as opposed to raw data packing. On the HP's, we notice a decrease in throughput of roughly 30% for each test, and on the SGI's we approximate a 50% to 60% decrease.

4 Improvements and future work

The data included here represents our initial results as we begin the process of characterizing the performance of parallel programs in distributed systems.

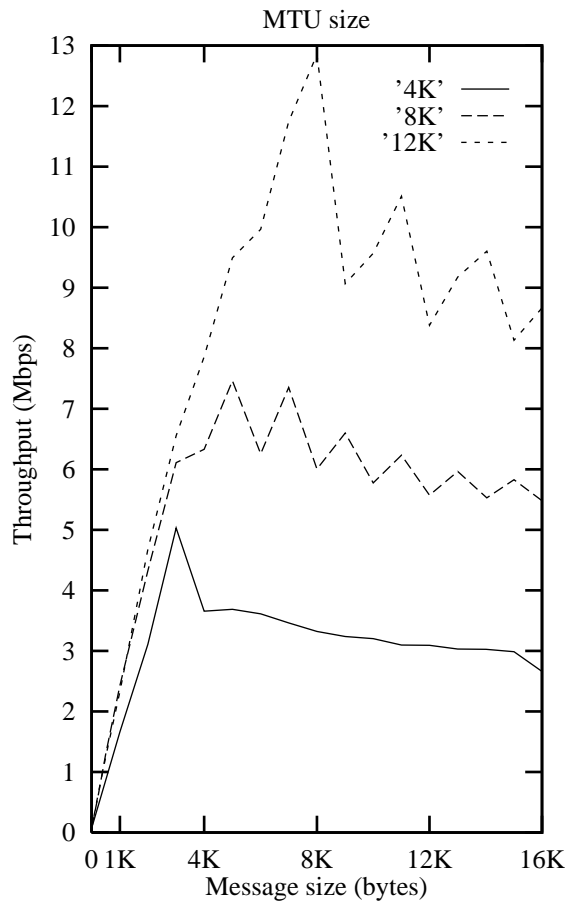


Figure 10: *Blast* test, TCP routing.

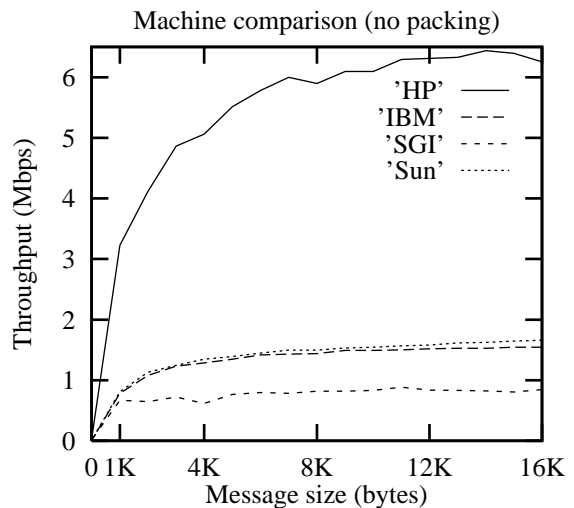


Figure 11: *Ping Pong* test, *pvmd* routing.

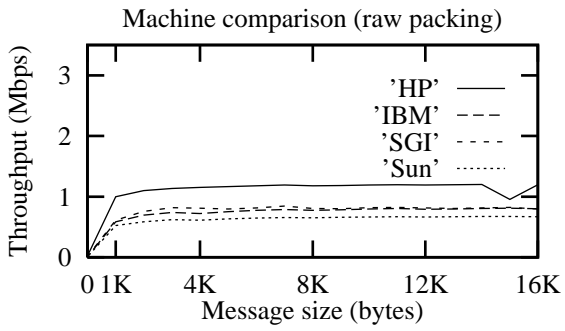


Figure 12: Ping Pong test, pvm routing.

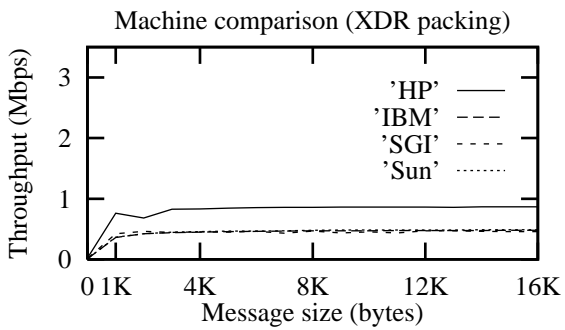


Figure 13: Ping Pong test, pvm routing.

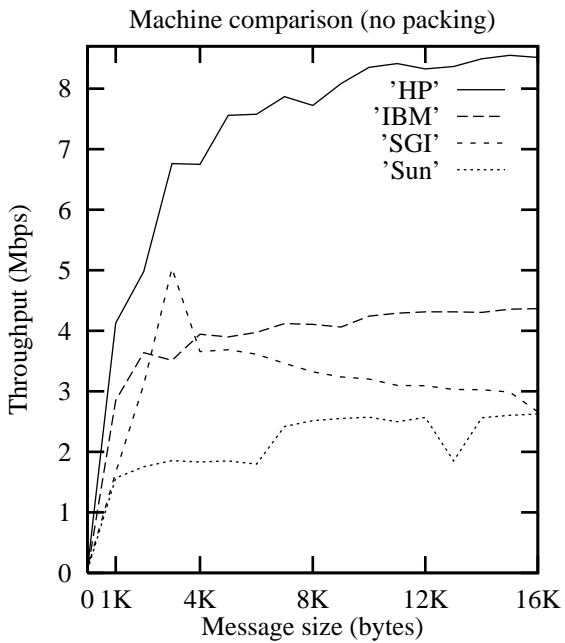


Figure 14: Blast test, TCP routing.

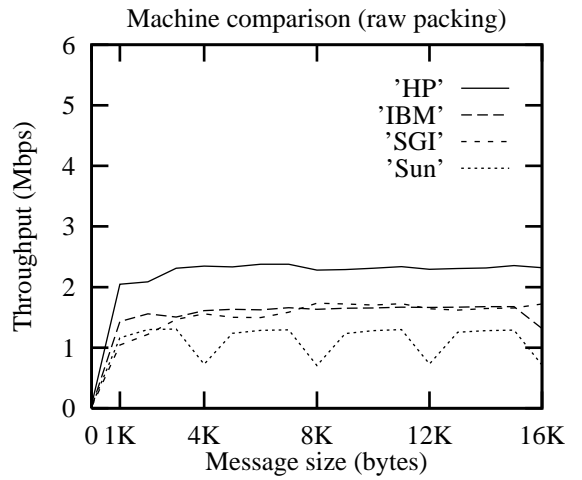


Figure 15: Blast test, TCP routing.

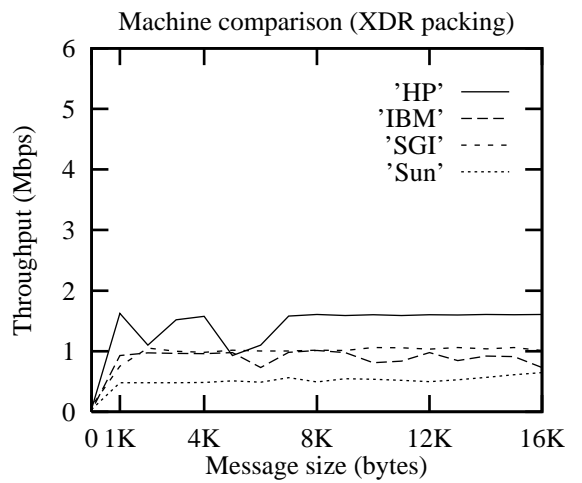


Figure 16: Blast test, TCP routing.

In future work, we plan to 1) investigate the effects of PVM's `PvmDataInPlace` packing, 2) run tests for a wider range and finer granularity of message sizes, particularly messages smaller than 1K, 3) utilize the full extent of the HEAT network, 4) repeat the tests on alternate network hardware platforms including a ForeSystems ATM switch, and 5) more accurately simulate the performance of real parallel applications by incorporating local computation time and a wider range of task configurations into `pvmtime`. Many of these results will be reported in [5].

5 Conclusions

We observed that the `pvm` can become a bottleneck in communication intensive applications, especially those in which multiple tasks reside on the same machine. Packing and unpacking messages also significantly limits throughput. Simple alterations to PVM applications, such as the use of TCP direct routing and a larger PVM MTU size show promising performance improvements, resulting in a throughput of up to 13 Mbps. However, it seems clear that PVM's efficiency must be further improved in order to better exploit the speed of the network hardware.

References

- [1] H. Chen, J. Hutchins, and J. Brandt. DEC FDDI Gigaswitch Performance. Technical Report in preparation, Sandia National Labs, Livermore, CA.
- [2] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam. PVM 3 User's Guide and Reference Manual. Technical Report ORNL/TM-12187, Oak Ridge National Labs, Oak Ridge, TN, May 1993.
- [3] Brian K. Grant and Anthony Skjellum. The PVM System: An In-Depth Analysis and Documenting Study - Concise Edition. Technical Report TR UCRL-JC-112016, Lawrence Livermore National Laboratories, Livermore, CA, 1992.
- [4] R. E. Cline Jr., H. Chen, and J. Hutchins. Network Performance Requirements for a Distributed Computing Environment. Technical Report in preparation, Sandia National Labs, Livermore, CA.
- [5] M. Lewis. PVM Communication Performance in New Distributed Computing Environments. Technical Report in preparation, Sandia National Labs, Livermore, CA.
- [6] V. S. Sunderam. PVM: A Framework for Parallel Distributed Computing. *Concurrency: Practice & Experience*, 2(4):315-339, December 1990.