Network Subsystems in MPPs: Where Did All the Performance Go?

*Dorgival O. Guedes** *Larry L. Peterson*

Department of Computer Science
University of Arizona
Tucson, AZ 85721, USA
E-mail: {dorgival,llp}@cs.arizona.edu

June 1997

Abstract

In this work we describe our results on identifying the most important overheads in the network subsystem of massively parallel processors (MPPs), specially the Intel Paragon. We show how poor implementation techniques currently prevent performance of applications using TCP/IP protocols to communicate between supercomputers connected by high performance networks from achieving data rates close to the capacity of the medium.

We identify some of the possible solutions to the problem, and discuss what can be expected in future systems. In particular, we present our results on evaluation of some of those solutions, including a user-level protocol implementation which scales with the number of concurrent connections and performance superior to the current system.

Resumo

Neste artigo apresenta-se resultados sobre a identificação dos principais gargalos em subsistemas de rede de processadores massivamente paralelos, abordando particularmente o Intel Paragon. Demonstra-se como técnicas de implementação correntes impedem que aplicações que utilizem protocolos como TCP/IP entre supercomputadores interligados por redes de alta velocidade alcancem taxas de transferência próximas à capacidade do meio de transmissão.

Algumas possíveis soluções para o problema são sugeridas, e discute-se o que pode ser esperado de sistemas futuros. Em particular, resultados de avaliação dos benefícios de algumas técnicas são apresentados, destacando-se uma implementação de protocolos no espaço da aplicação que apresenta performance e capacidade de expansão para multiplas conexões superior aos sistemas atuais.

*Sponsored by Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), Brazil, Process no. 200861/93-0

1 Introduction

Over the last few years the developments in the area of massively parallel processors (MPPs) have shown that we are now capable of building large scalable systems with peak performance in the TeraFLOPS range. Such machines are based on the use of high performance commodity processors connected by a high speed proprietary interconnection network. Examples of such systems are the IBM SP systems [Ger95] and the Intel Paragon [Int91].

What guarantees the high performance of such systems is the availability of interprocess communication primitives (IPC) which allow processes in different nodes of the machine to exchange information at very high rates with minimum delays. Whether the system provides a message passing interface or a distributed shared memory model, applications where hundreds or even thousands of nodes work together in a task are possible.

Now the local and wide area network technologies are starting to reach speed levels comparable to those of some of the current supercomputer interconnects, in the Gigabit per second range. These developments make it possible for us to think of building our own supercomputers by connecting our own workstations with a fast network. On the other hand, we can also imagine connecting our already powerfull MPPs to others in other laboratories. The need for such connection comes from the ever increasing need to share information among different sites and different computers, as well as from the need to work on even larger problems.

But although we have very high performance supercomputers and very fast network interconnects available today, like HiPPI [TR93] and ATM [Vet95], there has been only limited success on putting them together. In the application realm, one would expect to be able to extend a parallel application across multiple computers by using some already designed communication package, like PVM[Sun92] or MPI[GLS95], and some stable protocol suite like TCP/IP[1]. But things are not that simple.

The problems begin when we try to use any TCP/IP based code to push data over a connection between two MPPs linked by some high speed network. The user would be shocked by the low bandwidth achieved in most cases. Many solutions today tend to adopt specialized approaches [Dun96], which unfortunately do not scale well for a more general network. For example, the use of raw HiPPI connections is common in the supercomputer world, but such technique is limited to local HiPPI networks. When we intend to connect systems at different sites, separated by possibly multiple networks with different technologies, TCP/IP is still the easiest way to go[Com95].

But where does the performance go? A TCP/IP connection between two Intel Paragon

[1] Although an implementation of PVM in the Paragon may use its internal IPC for communication between nodes, communication with other machines must rely on TCP/IP to cross the external network

over a HiPPI switch, for example, usually gets just about two percent of the available bandwidth, pushing data at 2 MB/s, when the network bandwidth is 100 MB/s. In this work we intend to discuss what the problems are today that prevent us from using all the available bandwidth of such systems, and to present some results achieved by applying techniques to avoid them.

The rest of this paper is organized as follows: In the next section we present numbers to show the mismatch between the performance achieved between two nodes inside an MPP and between nodes in different computers over TCP/IP. By doing that we can identify points where performance drops occur, and try to identify the reasons for that. After that we discuss some of the techniques which can be used to avoid those problems, and present some results of our work which improved the overall performance. Finally, we present some conclusions and discuss some other options for future work.

2 The Network Subsystem in Current MPPs

As mentioned earlier, today supercomputers are based on the interconnection of high performance commodity processors using a high speed interconnect. Such interconnect may be a switched network (IBM SP), tree-based networks (Thinking Machines CM-5), a regular mesh (Intel Paragon), or some other design. The important aspect is that the programmer has direct access to communication primitives which make full use of the network capacity. In this research we focus on the Intel Paragon as a reasonable example of such a system, but most of the discussion presented here can be easily applied to others.

2.1 The Paragon Mesh

The compute nodes in the Intel Paragon are composed of two or three Intel i860 processors in each node, coupled with a custom designed mesh connection chip, the mesh processor, which is responsible for routing messages in transit in the mesh and to handle messages from or to that node. It has a five input crossbar switch, connecting the processors in each node to the four immediate neighbor nodes. Each connection link is capable of carrying data at 160 megabytes per second (MB/s).

One of the i860 in a node is the message processor, responsible for handling messages from and to the node, interfacing to the mesh processor. The remaining i860 (one or two, depending on the board model) are available for the application. Communication between the application processor(s) and the message processor is performed through shared memory structures, and is controlled by the interprocess communication (IPC) interface, called NX [Int93].

The operating system on each node can be either Intel's OSF/1, derived directly from the Mach operating system [RBG+93, RBF+89], or SUNMOS, a light weight kernel

developed at Sandia Labs [SS94]. OSF/1 provides the application running on any node with a complete interface to all system resources, while SUNMOS offers only a limited interface, allowing nodes in an application little more than message exchange, but with higher performance, due to the smaller overhead. That limited interface does not allow applications to access most devices, including external network interfaces, so SUNMOS is of little use if we intend to write network aware applications.

The Paragon systems available for our use at the University of Arizona and at Caltech were configured to run with OSF/1 only, so all numbers we got are for that system. When results for SUNMOS differ in a noticeable way we mention numbers from the literature.

2.2 Interprocess communication through the mesh

Application programs can make direct use of NX primitives to exchange data, or use special libraries which in turn sit on top of NX. One such library is the PVM package, for example. Since in this work we are interested in the maximum performance possible, we focus on the NX primitives directly.

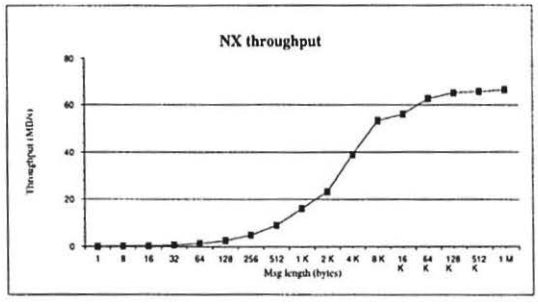

Figure 1: NX performance

Figure 1 shows the throughput achieved by an application transfering data between two nodes in messages of various sizes using NX primitives. The maximum rate achieved in this test, 66 MB/s, is well below the link capacity, so here we find the first point where performance is lost. The problem here is the operating system overhead. OSF/1 (and Mach, for that matter) has a lot of bookkeeping to do when handling memory around, and message sending and receiving requires message buffers to be allocated, filled and passed between the application processor and the message processor. All this limits the maximum throughput.

It is easy to show that OSF/1 is the culprit here by comparing those numbers with measurements of the same transfer under SUNMOS as mentioned in [Dun94]. Under

SUNMOS the transfer reaches up to 150 MB/s, very close to the link capacity. Obviously the overhead of OSF/1 limits the bandwidth between the application processor and the message processor. Nevertheless, OSF/1 offers many advantages over SUNMOS, like a much richer interface to operating system services, which justifies its use.

It is worth mentioning that although OSF/1 limits the maximum bandwidth of a single connection, it does not limit link utilization by connections from different nodes which happen to be routed through a common link. In that case all the capacity of the link is available (160 MB/s), and multiple transfers may take place concurrently, sharing the link capacity.

2.3 Raw HiPPI

From the previous section, it is clear that we cannot use all the bandwidth of a HiPPI channel from a compute node if the message has to travel the mesh to reach the network interface, which is usually the case. But can we get even close to those 66 MB/s? In order to answer that we must first determine the maximum throughput possible through the Paragon HiPPI interface. The experiment used to verify that was performed by running an application to push data between the nodes containing each endpoint of a HiPPI link. In this way, data from one node to the other had to go only from the application processor in one node to the HiPPI interface, across the cable and into the other node.

That is not exactly what happens when a compute node sends data to the HiPPI interface, though. In our test, data goes from an application processor to the interface. In the case of a compute node sending data, that data would go from the compute node, into the mesh, then to the mesh processor in the I/O node, which would store the data in memory. Finally the message processor in turn handles it to the interface. Nevertheless, out test should show a close upper bound to the maximum possible rate. In order to be exact, though, we must also repeat the same test moving the endpoints of the data transfer from the nodes containing the HiPPI boards to two compute nodes inside the mesh. The throughput measured that way would be the actual maximum throughput available for an application using the HiPPI interfaces, and would identify the overhead due to remote device access [FGB91].

Figure 2 shows the rates achieved by both the connection between I/O nodes and the connection between generic compute nodes.

In the I/O node case the maximum rate is 24 MB/s, already well below NX maximum throughput. In this case again the problem is due to operating system overhead. Although the application executes in the same node where the interface is located, each message to be sent must be placed in a Mach message and passed to the OSF/1 (Mach) kernel in that node for processing and delivery to the interface. The kernel must perform some security tests to verify if the message represents a valid request, and all this adds a lot of overhead

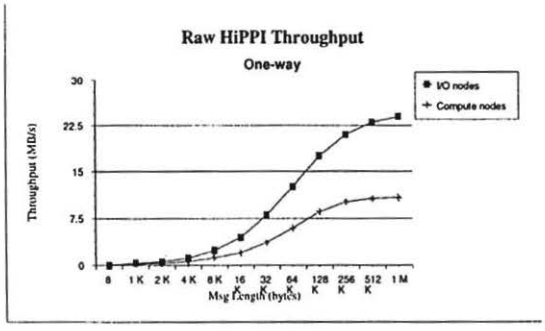

Figure 2: Raw HiPPI performance

in the case of Mach. Finally, data must be moved between the kernel and the interface. In many cases, when the data arrives before the application is able to define a buffer to hold it, additional copies are needed to move it from the interface or mesh processor to a system buffer, and later from that buffer to the one provided by the application. All those costs add up noticeably.

When we consider the generic compute node, though, the situation is even worse: the maximum bandwidth is limited to 11 MB/s now, already around just ten percent of the channel capacity. Here we see costs piling up. The raw HiPPI interface available to the application programmer is based on Mach NORMA IPC messages, messages used by Mach kernels in a cooperative network to communicate [Bar91]. When the application in the compute node has to send a message it issues a system call to deliver the message to the kernel in its own node, which causes the first overhead. Then the message is authenticated, encapsulated in a Mach NORMA message, and shipped over the mesh to the node containing the HiPPI interface. There it is copied from the mesh interface to memory, processed by the OSF/1 kernel in that node, and finally copied to the interface. The processing during delivery is similar.

2.4 Good old TCP/IP

But programming using raw HiPPI calls poses two problems for the application developer: First, it is a new API which has to be mastered. It would be easier if we could just use all the code already developed for more common network stacks, like TCP/IP. Second, raw HiPPI use is limited to computers connected to a single HiPPI domain. It does not provide any routing information across domains, no error control in case routers along the way drop a packet, and cannot be passed on to other kinds of networks. We need a more flexible transport protocols, which can take care of dropped packets, routing, and which is supported over different network mediums. Currently, no protocol suite can do

it better than TCP/IP.

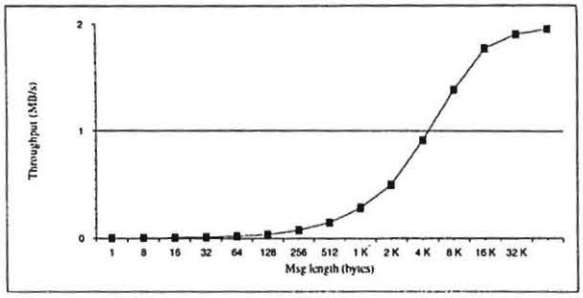

Figure 3: OSF/1 TCP performance

Results appear in figure 3 for the transfer of large data sets (8 MB) using various message sizes using OSF/1 TCP/IP. To understand what happens to make performance so low compared to the previous numbers, we must note three points: Segment size, protocol stack implementation and protocol processing cost.

First of all, the standard for TCP/IP implementation over HiPPI networks state that the maximum segment size allowed in this case is a little under 64 Kbyte [Ren97]. So we cannot benefit from sending data as huge 1 MB packets, as in previous tests. This limitation makes the packetization overhead (headers, HiPPI burst transfer setup and tear down) much higher, reducing the maximum achievable rate. For the compute node raw HiPPI test we got approximately 6 MB/s for 64 KB messages, so we cannot expect TCP to exceed that. But our results are still much lower, with a maximum of just about 1.96 MB/s.

Next factor to justify such low numbers is the structure of the protocol stack implementation, represented in figure 4. As a Mach derivative, Intel OSF/1 implements all protocol processing functions inside the Unix server, a process running in one of the nodes of the MPP, called the service node [LR94]. The reason for that is to simplify the management of host wide protocol information, like ports and sequence numbers. Using a centralized server, all this information is kept in one place. The problem is that such solution has two drawbacks: It adds an extra hop for every message, and an expensive one, since a Mach message has to go from the application address space, to the local kernel, to the kernel in the service node, and finally to the Unix server address space. Besides that, there is also the fact that the Unix server becomes a bottleneck when multiple nodes start concurrent TCP connections through the network interface. All data from all connections has to be processed in a single node, by a single process. That restricts performance.

Finally, there is the protocol processing cost itself. Every data in a TCP connection

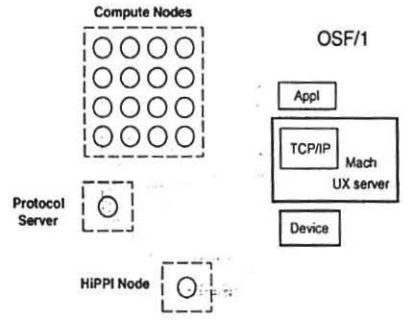

Figure 4: The Intel Paragon network subsystem

has to go through a checksum computation process. Since there is no hardware support for that in the interface, it requires data to cross the memory bus twice, and requires that computation be performed in the CPU for every byte transfered.

3 Where did it go?

So that is the bad news. Although a programmer can expect transfer rates on the order of tens of megabytes per second when communicating between nodes in an MPP, the effective rate available to transfer data to other computers across a fast network as HiPPI is lower than two megabytes per second. The main overhead factors were identified as:

System call overhead: Mach message construction and domain boundary crossings add a lot of overhead, as seen by comparing OSF/1 and SUNMOS NX performance.

Remote node system calls: Worse yet, system calls issued by applications in one node which have to be processed by another, like writing to a remote device or opening a TCP connection require many boundary crossings, with high overhead.

Centralized protocol processing: The use of the Unix server as the responsible for all TCP processing adds extra messages and creates a bottleneck to concurrent connections.

All-software protocol processing: Computing checksums in software, for example, create CPU overhead.

Once all these points have been identified, we can start to address the problems they represent.

4 How can we improve it?

There are solutions already proposed to some of the problems mentioned earlier, while others still require further investigation. In this section we sumarize some of the research in the area, and present our results in studying some of the topics.

4.1 Operating system structure

The difference in IPC performance between SUNMOS and Intel OSF/1 points clearly to the first point to be addressed: OSF/1's Mach heritage. Although it was an important project, with many contributions, Mach structure poses problems due to the high overhead it requires to allow applications to access systems resources, both on the local node and, more severely, on remote nodes [MB92]. SUNMOS can be pointed out as a possible solution when applications require just intra-MPP communication, but the lack of access to other system services like remote devices prevents it from being used when inter-MPP communication is a must.

There has been some research in the last few years pointing out that microkernel designs need not add such high overhead to system services [Lie95], and such results may eventually be retrofitted into the MPP arena. For example, the next generation Intel supercomputer [Cor] does not use the Mach kernel in the compute nodes, replacing it by a lighter, faster kernel, yet maintaining access to system resources for all nodes.

4.2 Integration of primitives

Comparisons between NX bandwidth and Mach NORMA IPC show how different performance is between restricted, application level limited communication and the more powerfull and secure inter-kernel communication schemes. Systems should be designed so that all levels could benefit from the faster technique. In particular, Mach NORMA IPC is a general solution, designed to work both on the MPP realm as in wide area networks connecting dispersed Mach machines. Tuning the inter-kernel communication and device access primitives to the features of the proprietary interconnect (and extending the interconnect with features like hardware based authentication and access control) would certainly yield faster systems [TB94].

We devised an experiment to try to identify how much Mach NORMA IPC limits the overall connection bandwidth: In stead of writing a simple data transfer program to be run in the compute nodes which would use the raw HiPPI device interface provided by OSF/1 (what would require Mach NORMA IPC messages between compute nodes and the I/O node with the actual device) we ran a distributed program composed of two processes, one executing in the I/O node and the other in the compute node. In such setup, the process in the compute node would not use raw HiPPI directly, but would

instead use NX to transfer the data to the process in the I/O node. That process in turn would execute a local system call to write to the interface.

In this case, we would expect the final performance for the transfer between two compute nodes to get closer to that achieved when the transfer occurred between I/O nodes, since the NX used from compute node to I/O node should not be a limiting factor. Our results, unfortunately, proved to be no better than the original results from figure 2, staying withing 10 percent of the compute node numbers. A more carefull analysis of the program and its interaction with the kernel provided some explanation. Although we gained in throughput by using NX in stead of NORMA IPC between compute node and I/O node, we inserted an application program in the I/O node, adding domain crossings from kernel to the application program and back. Besides that, in the NORMA IPC case, messages went from the the mesh processor directly to the message processor which performed all system functions, while our implementation had messages being processed by the mesh processor, then the message processor, which relaied them to the application processor, and then back to the message processor for operating system services.

Under such considerations, we can argue that the fact that we could at least achieve performance similar to that of the Mach NORMA IPC case is in fact promising, since we are dealing with much higher overheads in the I/O node itself. If we could retrofit our program which received NX messages and issued system calls into the kernel of the I/O node, we would most likely have got the performance gains we expected.

4.3 Avoid centralized servers

For an MPP, even more important than the performance of a single data transfer is how it behaves when operating with other connections in parallel. In order to determine that, we ran a set of experiments in which we started TCP connections between varying numbers of compute nodes in two separate Intel Paragons. We focused only on 64 KB messages, since we were interested in maximum throughputs. Figure 5 show our results.

It is clear how big a bottleneck the Unix server becomes. Although many of the processing costs are due to mesh transfers and to costs in the compute nodes, adding extra connections does not increase the aggregate throughput by much, reaching a peak at 2.2 MB/s. In fact the server does not scale at all after that since cache and buffer conflicts in the server actually cause performance to degrade as new connections are added. As one would expect, the throughput of each individual connection drops quickly as more connections have to share the limited bandwidth.

We have focused on this problem in our research, where we have succeeded in avoiding the Unix server bottleneck. The idea we have implemented is illustrated in figure 6, and is based on the user level implementation of protocol stacks as proposed by Maeda and Bershad [MB93].

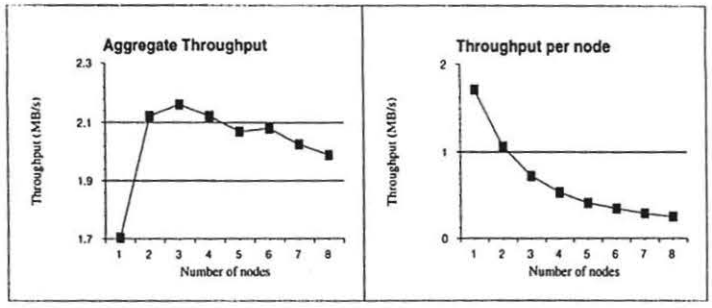

Figure 5: Performance of OSF/1 TCP for concurrent connections

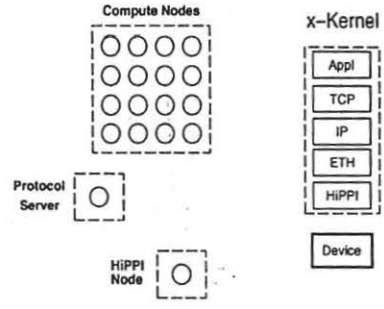

Figure 6: The user level implementation of the protocol stack

We have ported the x-kernel [OP92] to the Intel Paragon application address space and used it as the implementation framework to build a user-level TCP/IP protocol stack, complete with the TCP extensions for high bandwidth networks such as HiPPI. Our implementation makes use of Intel p-threads libraty [Int93] and the Mach out-of-kernel device interface [FGB91] to access the HiPPI board. With each node being able to process its own TCP connections, the Unix server is removed from the data path completely. The server is accessed only during connection setup and tear down, when the host wide context of TCP has to be updated. Once the connection is set, data moves directly between the I/O node and the compute node holding the process using each connection. This is possible by programming the packet filter at the I/O node [YBMM94] with information to identify the end point of each TCP packet. A more detailed discussion of our work can be found in [GP97].

At this point we make use of Mach NORMA IPC to reach the I/O node. As we have discussed, altering the kernel in that node to accept NX requests directly might improve

performance even further.

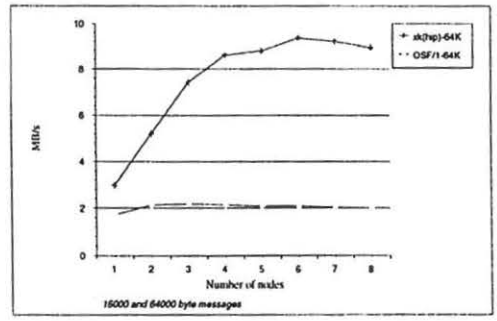

Figure 7: Aggregate throughput

We have performed the same connection scalability tests we used before for the OSF/1 protocol stack. Figures 7 and 8 show how our implementation compare to the that system. Clearly, our system achieves much better scalability, increasing the aggregate throughput up to 9.2 MB/s, much closer to the maximum of 12 MB/s achieved between I/O nodes for 64 KB messages.

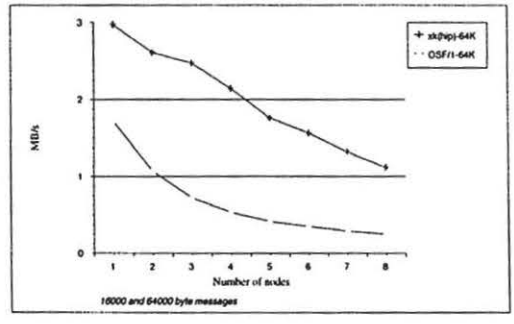

Figure 8: Throughput per node

Our solution scales well, increasing aggregate throughput for up to 6 concurrent node connections, after what performance of each connection begins to drop as all have to share the limited bandwidth. Such results show a great improvement over the current implementation.

4.4 Hardware assisted protocol processing

From figure 7 we can see that we have improved the performance of a single TCP connection by about 50 percent, but the maximum rate of 3 MB/s is still well below the

maximum rate achieved for a raw HiPPI connection between compute nodes, which was around 6 MB/s. To determine which were the limiting factors for a single connection, we had to investigate the protocol processing which now was taking place in the compute nodes. We did that by comparing the usual TCP connection performance with that of an altered TCP stack without checksum computation and with that of an UDP transfer with similar characteristics.

For 64 KB messages, TCP without checksum computation was able to reach 4.5 MB/s, another 50 percent gain. That shows that checksum computation is an important cost in the protocol stack operation. Furthermore, the UDP connection did not improve significantly over that: Maximum UDP throughput was 4.7 MB/s, showing that the remaining TCP processing costs were not as significant. The difference between the UDP numbers and the raw HiPPI numbers is due to unavoidable protocol processing costs, like demultiplexing, and due to the intrinsic overhead associated with the framework run time support, like p-thread context switching and system call processing.

That confirms what has been suggested by others: TCP/IP implementations in high performance systems can bennefit heavily from some kind of hardware assisted checksum computation associated with the network interface [Ste94]. One of the reasons such support is not present in the Paragon board is that its HiPPI interface was developped having in mind local access to HiPPI disks and other devices, not internetworking. Our results suggest that for future implementations such features should be considered.

4.5 Special-purpose protocols

In all our scalability tests we have used TCP as our transport protocol, mainly because of its widespread availability and its connection oriented service. But TCP was developed having in mind the Internet world, where connections operate independently, competing with others with completely different needs along its route. Its flow control techniques were all developed having such situation in mind.

In the MPP world, we will most likely need several concurrent connections from nodes in one computer to nodes in another one to carry out some computation[2]. All those connections will be competing with each other along the way, and their flow control mechanisms will be adjusting each one accordingly, resulting in the reduced per node rates seen in figure 8. But the flow control computation for each connection has no information about the cooperative behavior of all those connections: They should not compete with each other, but instead could cooperate in some way in order to achieve the best use of the medium. For example, they could use some sort of scheduling among themselves to avoid conflicts due to packets from different nodes arriving at a given router

[2]The option to this would be to use one process in each MPP as a centralizer for all communication, which would repeat the problems of the centralized protocol processor in the Unix server

or switch too close together.

That is the problem we intend to attack now. There are no results in this area yet, but we hope our research can provide some answers about what can be achieved by making the cooperatiove nature of the connections explicit.

5 Conclusions

We have shown how the high throughput rates available for communication among nodes in a single MPP get reduced as we add the steps and elements needed to allow simple communication with other computers, what usually implies a TCP/IP protocol stack. We were able to identify losses due to system call processing overhead, inter-kernel communication primitives, overhead due to message fragmentation requirements, centralized server processors and lack of hardware support for simple tasks.

Based on the problems identified some solutions were proposed and discussed to show how future network subsystem implementations can avoid the same limitations. Some of the techniques proposed are:

- Reduction of system call overhead in micro-kernels.
- Better integration of mesh access primitives and inter-kernel communication functions.
- Implementation of per-node protocol processing (user-level protocol stacks).
- Use of hardware assistance for checksum computation.
- Development of special protocols for cooperative connections.

In particular, we have presented our results with a user-level per node implementation of the TCP/IP protocol stack, which achieves higher throughput thant the original system, with good scalability with the number of nodes.

References

[Bar91] Joseph S. Barrera. A fast Mach network IPC implementation. In *Proceedings of the Usenix Mach Symposium*, 2560 Ninth Street, Suite 215, Berkeley CA 94710, November 1991. Usenix Association.

[Com95] Douglas E. Comer. *Internetworking with TCP/IP*, volume 1. Prentice Hall, 3rd edition, 1995.

[Cor] Intel Corporation. Intel TeraFLOPs supercomputer project home page. http://www.ssd.intel.com/.

[Dun94] Thomas H. Dunigan. Early experiences and performance of the Intel Paragon. Technical Report ORNL/TM-12194, Oak Ridge National Laboratory, oct 1994.

[Dun96] Thomas H. Dunigan. Performance of ATM/OC-12 on the Intel Paragon. Technical Report ORNL/TM-13239, Oak Ridge National Laboratory, may 1996.

[FGB91] Alessandro Forin, David Golub, and Brian Bershad. An I/O system for Mach 3.0. In *Proceedings of the Usenix Mach Symposium*, 2560 Ninth Street, Suite 215, Berkeley CA 94710, November 1991. Usenix Association.

[Ger95] Jerry Gerner. Input/output on the IBM SP2— an overview, 1995. Available at `http://www.tc.cornell.edu/SmartNodes/Newsletters/IO.series/intro`.

[GLS95] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI: Portable Parallel Programming with the Message Passing Interface.* MIT Press, 1995.

[GP97] Dorgival O. Guedes and Larry L. Peterson. Eliminating the network subsystem bottleneck in MPPs. To appear, jun 1997.

[Int91] Paragon XP/S product overview. Intel Corporation, 1991.

[Int93] Intel Corperation. *Paragon User's Guide*, oct 1993.

[Lie95] Jochen Liedtke. On micro-kernel construction. In *Proceedings of the Fifteenth ACM Symposium on Operating System Principles.* ACM, December 1995.

[LR94] John LoVerso and Paul Roy. The network architecture of OSF/1 AD version 2. In *OSF/RI Operating Systems Collected Papers Vol. 3.* OSF Research Institute, February 1994.

[MB92] Chris Maeda and Brian N. Bershad. Networking performance for microkernels. In *Proceedings of the Third Workshop on Workstation Operating Systems*, May 1992.

[MB93] Chris Maeda and Brian N. Bershad. Protocol service decomposition for high-performance network. In *Proceedings of the Fourteenth ACM Symposium on Operating System Principles*, December 1993.

[OP92] S. W. O'Malley and L. L. Peterson. A dynamic network architecture. *ACM Transactions on Computer Systems*, 10(2):110–143, May 1992.

[RBF+89] Richard Rashid, Robert Baron, Alessandro Forin, David Golub, Michael Jones, Daniel Julin, Douglas Orr, and Richard Sanzi. Mach: A foundation for open systems. In *Proceedings of the Second Workshop on Workstation Operating Systems(WWOS2)*, sep 1989.

[RBG+93] Paul Roy, David Black, Paulo Guedes, John LoVerso, Durriya Netterwala, Faramarz Rabii, Michael Barnett, Bradford Kemp, Michael Leibensperger, Chris Peak, and Roman Zajcew. An OSF/1 unix for massively parallel multicomputers. In *OSF/RI Operating Systems Collected Papers Vol. 2*. OSF Research Institute, Cambridge, MA, October 1993.

[Ren97] J. Renwick. IP over HIPPI. Request for Comments (Experimental) RFC 2067, Internet Engineering Task Force, January 1997.

[SS94] Subhash Saini and Horst D. Simon. Applications performance under OSF/1 AD and SUNMOS on Intel Paragon XP/2-15. In *Proceedings of Supercomputing'94*, Washington, DC, nov 1994.

[Ste94] Peter A. Steenkiste. A systematic approach to host interface design for high-speed networks. *Computer*, 27(3):47–58, March 1994.

[Sun92] Vaidy Sunderam. Concurrent computing with PVM. In *Proceedings of the Workshop on Cluster Computing*, Tallahassee, FL, December 1992. Supercomputing Computations Research Institute, Florida State University. Proceedings available via anonymous ftp from ftp.scri.fsu.edu in directory pub/parallel-workshop.92.

[TB94] John Michael Tracey and Arindam Banerji. Device driver issues in high-performance networking. In *Proceedings of the 1994 USENIX Symposium on High-Speed Networking*, August 1994.

[TR93] Don Tolmie and John Renwick. Hippi: Simplicity yields success. *IEEE Network*, January 1993.

[Vet95] Ronald J. Vetter. Atm concepts, architectures and protocols. *Communications of the ACM*, 38(2), February 1995.

[YBMM94] Masanobu Yuhara, Brian N. Bershad, Chris Maeda, and J. Eliot B. Moss. Efficient packet demultiplexing for multiple endpoints and large messages. In *USENIX Conference Proceedings*, pages 153–165, San Francisco, CA, Winter 1994. USENIX.