1994

A Communication System for High-Performance Distributed Computing

Salim Hariri
Syracuse University

JongBaek Park
Syracuse University

Manish Parashar
Syracuse University

Geoffrey C. Fox
Syracuse University

A Communication System for High-Performance Distributed Computing

Salim Hariri* [1], JongBaek Park, Manish Parashar and Geoffrey C. Fox
Northeast Parallel Architectures Center
Syracuse University
Syracuse, NY 13244
hariri@cat.syr.edu

Abstract

With the current advances in computer and networking technology coupled with the availability of software tools for parallel and distributed computing, there has been increased interests in high-performance distributed computing (HPDC). We envision that HPDC environments with supercomputing capabilities will be available in the near future. However, a number of issues have to be resolved before future network-based applications can exploit fully the potential of HPDC environment. In this paper, we present an architecture of a high-speed local area network and a communication system that provides HPDC applications with high bandwidth and low latency. We also characterize the message-passing primitives required in HPDC applications and develop a communication protocol that implementes these primitives efficiently.

1 Introduction

Decades of "experimentation" with parallel and distributed computing has established the importance of handling real-world applications. Enormous amount of research is being invested into exploring the nature of a general, cost-effective, scalable yet powerful computing model that will meet the computational and communication requirements of the wide range of applications that comprise the Grand Challenges (climate modeling, fluid turbulence, pollution dispersion, human genome, ocean circulation, quantum chromodynamics, semiconductor modeling, superconductor modeling, etc.). Based on these premises, there has been increased interests in high-performance distributed computing (HPDC). We envision that an HPDC environment with supercomputing capabilities will be available in near future. The driving forces towards this end are (1) the advances in processor technology, (2) the emergence of high-speed networks (3) the development of software tools and programming environment.

Current workstations are capable of delivering tens and hundreds of Megaflops of computing power; for example, a cluster of 1024 DEC Alpha workstations provides a combined computing power of 150 Gigaflops, while the same sized configuration of the CM5 from Thinking Machines Inc. has a peak rating of only 128 Gigaflops [16]. Thus, aggregate computing power of a group of high-performance workstations can be comparable to that of supercomputers. Further, workstations are general-purpose, flexible and cost-effective; the cost-performance ratio for a workstation today is about 5000 peak flops/$ while that for a conventional supercomputer like a Cray is only 500 to 1000 peak flops/$ [16]. Furthermore, it has been shown that the

[1]* Author to whom all correspondence should be sent

average utilization of a cluster of workstations is only around 10% [17]; most of their computing capacity is sitting idle. This un-utilized or wasted fraction of the computing power is sizable and, if harnessed, can provide a cost-effective alternative to supercomputing platforms.

Advances in computer networking technology have introduced high speed, reliable networks capable of providing high transfer rates. Current trend in local area networks is towards higher communication bandwidth as we progress from Ethernet networks that operate at 10 Mbps (Megabit/sec) to higher speed networks such as Fiber Distributed Data Interface (FDDI) networks that operate at 100 Mbps. Furthermore, it is expected that soon these networks will operate in Gbps (Gigabit/sec) range.

Thus, it has been established that current clusters of high-performance workstations have the aggregate computing power to provide an HPDC environment that utilizes high speed networks (e.g., ATM, SONET, HUB-based LAN) [15]. It has also been established that it is not cost-effective to introduce new parallel architectures to deliver the computing power. Consequently, we envision that future computing environments need to capitalize on and effectively utilize the existing heterogeneous computing resources.

A number of issues have to be resolved to exploit the full potential of processing and networking technologies. The primary barrier in building HPDC environment lies in the limited communication bandwidth available at the application level. In current local area networks (LANs), the bandwidths achievable at the application level are often an order of magnitude lower than that provided at the network medium [12, 13]. For example, out of the physical bandwidth of 10 Mbps available at the medium level of the Ethernet, only about 1.2 Mbps is available to the application [12]; it is therefore not sufficient to have even a Gigabit data link if user applications could only use a small portion of the bandwidth. This degradation in performance occurs because of two main reasons: (1) Host-to-network interface which is characterized by its excessive overhead of processor cycles and system bus capacity, and because of heavy usage of timers, interrupts, and memory read/writes; and (2) the standard protocols that are implemented as a stack of software layers and consume most of the medium capacity and provide very little bandwidth to the application.

In this paper, we present an approach to provide an efficient communication environment for High-Performance Distributed Computing (HPDC). The main objectives of this research are:

(1) To develop a high-speed local area network (HLAN) architecture for HPDC. The HLAN consists of high-performance computers and high-speed networks. By employing two-tiered stack of protocol architecture, the HLAN architecture supports two modes of operation: normal-speed mode (where standard protocols are used) and high-speed mode (where high-speed protocols are used). In this paper, we show two examples to implement the HLAN architecture. The first one is based on ring network with a host interface processor and the other one is based on the ATM network.

(2) To develop a message passing interface for the HPDC environment. We first study some pallel and distributed software tools that provide message passing primitives and then identify a maximal set of primitives for the proposed HPDC environment.

(3) To develop a high-speed communication protocol (HCP) which provides an efficient communication environment suitable for HPDC. HCP is characterized with its simple communication scheme to provide low latency to operate in the Gbps range, and concurrent communication capability to allow multiple processes to communicate in parallel over the network.

3 Software Support for HPDC

The approach to identify the message passing primitives is carried out in two steps: 1) analyze the message-passing primitives provided by existing software tools on current parallel and distributed systems; and 2) identify a maximal subset of message passing primitives that can be efficiently implemented by a communication protocol for parallel/distributed computing.

3.1 Characterization of Message Passing Primitives

In order to identify the message passing services for HPDC, we first study the primitives provided by some current parallel/distributed programming tools. The software tools studied include EXPRESS [21], PICL [25], PVM [26], ISIS [23], and the iPSC communication library [27]. These tools were selected because of their availability at the Northeast Parallel Architecture Center at Syracuse University and also the following two reasons: (1) they support most potential computing environments, i.e., parallel, homogeneous and heterogeneous distributed systems; and (2) they are either portable tools (EXPRESS, PICL and PVM) or hardware dependent tools (the iPSC communication library). There is an increased interest in the standardization of message-passing primitives supported by software tools for parallel/distributed computing [28]. The characterization provided in this section can be viewed as step in this direction. The communication primitives supported by existing libraries can be characterized into five classes, viz., point-to-point communication, group communication, synchronization, configuration/control/management, and exception handling.

Point-to-Point Communication

The point-to-point communication is the basic message passing primitive for any parallel/distributed programming tools. To provide efficient point-to-point communication, most systems provide a set of function calls rather than the simplest *send* and *receive* primitives.

- Synchronous and Asynchronous Send / Receive: The choice between synchronous and asynchronous primitives depends on the nature and requirements of the application. As a result, most tools support both, asynchronous and synchronous send/receive primitives. To provide asynchronous message processing, additional supporting functionality must be provided in the tools. For example, 1) poll/probe the arrival and/or information of incoming messages e.g., *extest* and *probe*, used in EXPRESS and PVM, respectively; 2) install a user-specified handler for incoming messages e.g., *exhandle* or *hrecv* used in EXPRESS or iPSC, respectively; and 3) install a user-specified handler for outgoing messages, e.g., *hsend* used in iPSC.

- Synchronous/Asynchronous Data Exchange: There are at least two advantages for providing such primitives. First, user is freed from having to decide which node should read first and which node should write first. Second, it allows optimizations to be made for both speed and reliability.

- Non-contiguous or Vector Data: One example of transferring a non-contiguous message is sending a row (or column) of a matrix that is stored in column-major (or row-major) order. For example, *exvsend/exvreceive* used in EXPRESS.

Group Communication

Group communication for parallel or distributed computing can be further classified into three categories, *1-to-many*, *many-to-1*, and *many-to-many*, based on the number of senders and receivers.

- 1-to-Many Communication: Broadcasting and multicasting are the most important examples of this category. Some systems do not explicitly use a separate *broadcast* or *multicast* function call. Instead, a wild card character used in the destination address field of point-to-point communication primitives, provides multicasting functions. It is important to note that in ISIS broadcast primitives with different types and order are available to users. Users can choose the proper broadcast primitives according to the applications.
- Many-to-1 Communication: In many-to-1 communication, one process collects the data distributed across several processes. Usually, such function is referred to as *reduction operation* and must be an associative, commutative function, such as, addition, multiplication, maximum, minimum, logical AND, logical OR, or logical XOR. For example, *g[op]0* and *g[type][op]* in PICL and iPSC, where *op* denotes a function and *type* denotes its data type.
- Many-to-Many Communication: There are several different types of many-to-many communications. The simplest example is the case where every process needs to receive the result produced by a reduction operation. The communication patterns of many-to-many operations could be regular or irregular.

Synchronization

A parallel and distributed program can be divided into several different computational phases. To prevent asynchronous message from different phases interfering with one another, it is important to synchronize all processes or a group of processes. Usually, a simple command without any parameters, such as, *exsync*, *sync0*, *gsync* in EXPRESS, PICL, and iPSC respectively, can provide a transparent mechanism to synchronize all the processes. However, there are several options that can be adopted to synchronize a group of processes. In PVM, *barrier*, which requires two parameters *barrier_name* and *num*, blocks caller until a certain number of calls with the same barrier name made. In PICL, *barrier0* synchronizes the node processors currently in use. In iPSC, *waitall* and *waitone* allow the caller to wait for specified processes to complete.

Another type of synchronization is that one process is blocked until a specified event occurred. In PVM, *ready* and *waituntil* provide event synchronization by passing the signal. In ISIS, the order of events is used to define *virtual synchrony* and a set of token tools (e.g., *t_sig, t_wait, t_holder, t_pass, t_request*, etc.) are available to handle it. In fact, event detection is a very powerful mechanism for exception handling, debugging, as well as performance measurement.

Configuration, Control, and Management

The tasks of configuration, control, and management is quite different from system to system. A subset of the configuration, control and management primitives supported by the studied software tools are such as to allocate and deallocate one processor or a group of processors, to load, start, terminate, or abort programs, and for dynamic reconfiguration, process concurrent or asynchronous file I/O, nad query the status of environment.

Exception Handling

In a parallel or distributed environments, it is important that the network, hardware and software failures must be reported to the user's application or system kernel in order to start a special procedure to handle the failures. In traditional operating systems such as UNIX, exception handling is processed by event-based approach, where a signal is used to notify a process that an event has occurred and after that a signal handler is invoked to take care of the event. Basically, an event could be a hardware condition (e.g., bus error) or software condition (e.g., arithmetic exception). For example, in the iPSC library, a user can attach a user-specified routine to respond to a hardware exception by the *handler* primitive. In ISIS, a set of *monitor* and *watch* tools are available to users. EXPRESS supports tools for debugging and performance evaluation. PICL supports tools for event tracing.

3.2 Message-Passing Primitives

Based on the characterization of message-passing techniques used in parallel/distributed computing presented in Table 1, we identify the set of primitives which can efficiently implement the primitives supported by most software tools for parallel and distributed computing. The services can be broadly classified as data transfer services, synchronization services, system management/ configuration services and error handling services. Data transfer services include point-to-point services for sending, receiving and exchanging messages and group communication services for broadcasting and multicasting data (*hcp_send, hcp_receive, hcp_exchange, hcp_bcast*). Synchronization services allow a processor to lock resources so that no other processor can access them(*hcp_barrier*). This service enables mutually exclude access of resources shared between processors. The *hcp_barrier* primitive enables a specified number of processor to synchronize at a logical barrier befo re proceeding. System management/ configuration services (*hcp_probe, hcp_msgstat, ...*) include calls to monitor sent and arriving messages, the current status of the network and hosts and to configure the hosts into logi cal groups and for adding/ deleting hosts from/to these logical groups. Special error handling services include the *hcp_signal* primitive which sends a high priority message to all hosts to propagate any error status and the *hcp_log/chkpt* primitive to enable checkpointing and logging of previously specified data for debugging purposes. When the *hcp_log/chkpt* signal is sent, all processors dump this data into a lo g file and proceed with their computation. In what follows, we describe how some of the services shown in Table 2 are implemented in HCP.

4 HCP Implementation Issues

In this section, we briefly describe the design of HIP to offload the protocol processing from the host and the operation of HCP.

4.1 Host Interface Processor

HIP is a communication processor capable of operating in two modes of operation such that either or both of these modes can be active at a given time. Figure 6 shows the block diagram of the main functional units of the proposed HIP. The HIP design consists of five major subsystems: a Master Processing Unit (MPU),

GA will read the frame. The FS field is used for acknowledgment; when a node receives erroneous frame, it sets the FS field and the source will respond accordingly. In order to prevent duplicate frames being received by a station, the one-bit field F/D is used to denote whether the received frame is new or duplicated.

4.4 Performance Results

We analyzed the performance of the proposed HLAN environment; in this paper, we summarize the main results and the detailed analysis can be found in [5]. The results show that the application-to-application bandwidth over the D-net is about 40–45 % of medium bandwidth, which is a significant improvement compared with the performance of standard protocol implementation (which provides application with a small fraction of medium speed, for example, about 10 % [12]). In addition, the networkwide bandwidth provided by the D-net, which we define as the ratio of the total number of bits transmitted over the D-net to the time taken for the transmission, can be even more than the medium bandwidth due to the concurrent communication capability of HCP and HIP. Furthermore, for the case where interprocess communication is directed only to neighboring node such that application processes are arranged to communicate in pipelined manner, we can obtain even better performance. The networkwide bandwidths for this case are approximately proportional to the number of computers; the more computers we use, the more parallel connections can be established. The networkwide bandwidth available for this case is much higher than the medium bandwidth provided. For example, for a network with 15 computers, the effective application-to-application transfer rates are approximately 690 Mbit/sec and 3.4 Gbit/sec for 100 Mbit/sec and 1 Gbit/sec channels respectively. We also obtained the broadcasting rate over the S-net to be more than 80 % of 100 Mbps channel and 35 % of 1 Gbps medium.

5 Application Example

In this section, we analyze the performance of the LU decomposition problem when it runs on the proposed HPDC environment. LU decomposition problem forms an integral part of many scientific and engineering problems to solve systems of linear equations takes advantage of the broadcast communication capability of HCP.

5.1 Algorithm

Host-node programming model can be used, where the host program partitions the input data and distributes it to node processors and collects the results from each node. We use the same node algorithm reported in [4, 6] for our analysis as shown below:

---

if $my\_num = 0$ /* for the first block */
    $Factorize$ block 0 and obtain result A;
    Broadcast A;
    $Update$ the remaining blocks;

For step $i = 1$ to $N \cdot n - 1$ /* for the subsequent blocks */
    if $my\_num = i \bmod N$

Figure 11

Factorization	Update
$\frac{b(b+1)(3(m-(i-1)b)-b)}{3}$	$b^3 + 2b^2(m - ib)$

Table 3: Number of floating-point operations at each step of the LU algorithm

Table 3 shows the number of floating-point operations per block at step i for each computer. Using this analysis, we can estimate the effective MFLOPS as follows:

$$MFLOPS_{LU} = \frac{(2/3) \cdot m^3}{T_{LU}} \tag{1}$$

where the numerator represents the approximate number of floating-point operations involved in whole computation, which can be approximated from Table 3 or can be found in [6]. Speedup performance can be represented as

$$Speedup_{LU} = \frac{(2/3) \cdot m^3 \cdot (1/MFLOPS_{computer})}{T_{LU}} \tag{2}$$

where the numerator is the single computer execution time.

The speedup and effective MFLOPS are shown in Figures 13 and 14. In this analysis, 4 computers are used with 20 and 40 MFLOPS respectively and channel bandwidths of 100 Mbps and 1 Gbps are assumed.

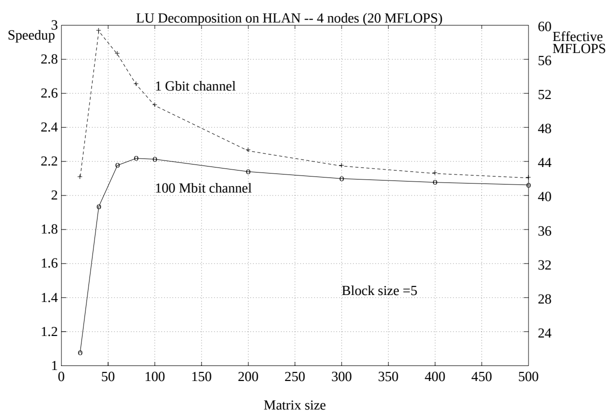

Figure 13: Effective MFLOPS for LU decomposition (20 MFLOPS computers)

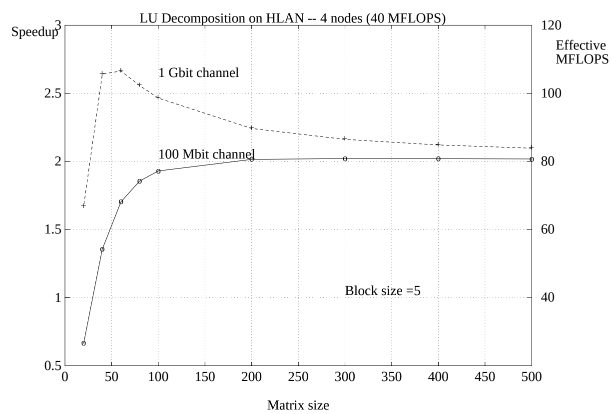

Figure 14: Effective MFLOPS for LU decomposition (40 MFLOPS computers)

6 Conclusion

The evolution of processor and networking technology has made the high-performance distributed computing (HPDC) attractive and cost-effective. We envision that high-speed network will allow its users to treat multiple computing resources as a single system rather than a network of computers.

In this paper, we proposed an architecture of high-speed local area network (HLAN) that capitalizes on the current advances in processor technology, software and networking technology. The HLAN supports two modes of operation: Normal-Speed Mode (NSM) where standard transport protocols are used to transmit and/or receive data over a channel allocated to this mode; and High-Speed Mode (HSM) where processes can access directly the HIP software layer to achieve the application-level transfer rates comparable to the medium speed.

We analyzed the primitives, supported by existing parallel and distributed software tools and characterize them into five categories; *point-to-point communication*, *group communication*, *synchronization*, *configuration / control / management*, and *exception handling*. Based on the analysis, we identified a set of primitives for the proposed message passing interface.

We also demonstrated the performance gain of an application example running on the HLAN and showed that the proposed HPDC environment is capable of providing supercomputing performance.

We are currently studying to implement the HCP protocol over the emerging ATM (Asynchronous Transfer Mode) networks incorporating the D-net, S-net and the normal speed network into one network.

References

[1] J. B. Park and S. Hariri, " Architectural Support for High-Performance Distributed Computing," *Proceedings of the 12th Annual IEEE International Phoenix Conference on Computers and Communications'93 (IPCCC-93)*, pp. 319–325, March 1993.

[2] S. Hariri, M. Parashar and J. B. Park, " Software and Hardware Support for Workstation-based Supercomputing," *Proceedings of the 26th Hawaii International Conference on System Science (HICSS-26), IEEE*, vol. 1, pp. 286–295, January 1993.

[3] S. Hariri, J. B. Park, F-K. Yu, and M. Parashar, and G. C. Fox, " A Message Passing Interface for Parallel and Distributed Computing," *Proceedings of the 2nd International Symposium on High-Performance Distributed Computing (HPDC-2), IEEE*, pp. 84–91, July 1993.

[4] M. Parashar, S. Hariri, G. Mohamed and G. C. Fox, " A Requirement Analysis for High Performance Distributed Computing over LANs, " *First International Symposium on High-Performance Distributed Computing (HPDC-1), IEEE*, pp. 142–151, September 1992.

[5] J. B. Park, *Ph.D dissertation: A Communication Protocol and Architectural Support for High-Performance Distributed Computing,* Syracuse University, 1993.

[6] A. G. Mohamed, G. C. Fox and G. Von Laszewski, " Blocked LU Factorization on a Multiprocessor Computer," Technical Report SCCS-94b, Northeast Parallel Architectures Center, Syracuse University, April 1992.

[7] M. J. Dayde and I. S. Duff, " Level 3 BLAS in LU Factorization on the CRAY-2, ETA-10P, and IBM 3090-200/VF, " *The International Journal of Supercomputer Applications,* Vol. 3, No. 2, pp. 40-70, Summer 1989.

[8] T. F. La Porta and M. Schwartz," Architectures, Features, and Implementation of High-Speed Transport Protocols," *IEEE Network Magazine*, pp. 14–22, May 1991.

[9] O. Menzilcioglu and S. Schilck, " Nectar CAB: A High-Speed Network Processor," *Proceedings of International Conference on Distributed Systems,* pp. 508–515, July 1991.

[10] H. Kanakia and D. R. Cheriton, " The VMP Network Adapter Board: High- performance Network Communication for Multiprocessors," *Proceedings of the SIGCOMM Symposium on Communications Architectures and Protocols,* pp. 175–187, August 1988.

[11] D. R. Cheriton and C. L. Williamson, " VMTP as the Transport Layer for High-Performance Distributed Systems," *IEEE Communication Magazine*, pp. 37–44, June 1989.

[12] G. Chesson, " The Protocol Engine Design," *Proceedings of the Summer 1987 USENIX Conference,* pp. 209–215, November 1987.

[13] D. D. Clark, M. L. Lambert and L. Zhang," NETBLT: A High Throughput Protocol," *Proceedings of SIGCOMM'87, Computer communications review,* Vol. 17, No. 5, pp. 353–359, 1987.

[14] C. Partridge," How Slow Is One Gigabit Per Second? " *Computer Communication Review*, Vol. 20, No. 1, pp. 44–52, January 1990.

[15] H. T. Kung, " Gigabit Local Area Networks: A Systems Perspective," *IEEE Communications Magazine,* pp. 79–89, April 1992.

[16] Gordon Bell," Ultra Computers : A Teraflop Before Its Time," *Communications of the ACM*, Vol. 35, No. 8, pp. 27–47, August 1992.

[17] P. Kruegeer and R. Chawla, " The Stealth Distributed Scheduler," *Proceedings of the 11th International Conference on Distributed Computing Systems,* pp. 336–343, May 1991.

[18] M.P. Papazoglu and P. E. Pintelas, " A Versatile Kernel Proposal for Multi- microprocessor System Environment," *Microprocessing and Microprogramming , North Holland,* pp.11–21, 1988.

[19] F. E. Ross, " An overview of FDDI: The Fiber Distributed Data Interface," *IEEE Journal on Selected Areas in Communications,* pp. 1043–1051, September 1989.

[20] D. Y. Cheng, " A Survey of Parallel Programming Tools," *Technical Report RND-91-005, NASA Ames Research Center,* Moffett Field, CA, May 1991.

[21] Express 3.2, Introductory Guide for Workstations, *Parasoft Corporation*, 1991.

[22] Express Fortran, User's Guide, *Parasoft Corporation*, 1990.

[23] K. Birman, R. Cooper, T. Joseph, K. Marzullo, and M. Makpanggou, The ISIS System Manual, 1990.

[24] K. Birman, A. Schiper, and P. Stephenson, " Lightweight Causal and Atomic Group Multicast," *ATM trans. Computer Systems,* Vol. 9, No. 3, pp. 272–314, August 1991.

[25] G. A. Geist, M. T. Heath, B. W. Peyton and P. H. Worley, " A User's Guide to PICL: A Portable Instrumented Communication Library," Technical Report ORNL/TM-11616, Oak Ridge National Laboratory, October 1991.

[26] A. Beguelin, J. Dongarra, A. Geist, R. Manchek and V. Sunderam, " A User's Guide to PVM: Parallel Virtual Machine," Technical Report ORNL/TM-11826, Oak Ridge National Laboratory, July 1991.

[27] Intel Supercomputer System Division, Beaverton, Oregon, " iPSC/2 and iPSC/860 User's Guide," 1991.

[28] D. W. Walker, " Standards for Message-Passing in a Distributed Memory Environment," Technical Report ORNL/TM-12147, Oak Ridge National Laboratory, August 1992.