1993

A Message Passing Interface for Parallel and Distributed Computing

Salim Hariri
Syracuse University, Northeast Parallel Architectures Center

JongBaek Park
Syracuse University, Northeast Parallel Architectures Center

Fang-Kuo Yu
Syracuse University, Northeast Parallel Architectures Center

Manish Parashar
Syracuse University, Northeast Parallel Architectures Center

Recommended Citation

Hariri, Salim; Park, JongBaek; Yu, Fang-Kuo; and Parashar, Manish, "A Message Passing Interface for Parallel and Distributed Computing" (1993). *Northeast Parallel Architecture Center*. 37.
https://surface.syr.edu/npac/37

A Message Passing Interface for Parallel and Distributed Computing

Salim Hariri, JongBaek Park, Fang-Kuo Yu, Manish Parashar and Geoffrey C. Fox
Northeast Parallel Architectures Center
Syracuse University
Syracuse, NY 13244

Abstract

The proliferation of high performance workstations and the emergence of high speed networks have attracted a lot of interest in parallel and distributed computing (PDC). We envision that PDC environments with supercomputing capabilities will be available in the near future. However, a number of hardware and software issues have to be resolved before the full potential of these PDC environments can be exploited. The presented research has the following objectives: (1) to characterize the message-passing primitives used in parallel and distributed computing; (2) to develop a communication protocol that supports PDC; and (3) to develop an architectural support for PDC over gigabit networks.

1 Introduction

The proliferation of high performance workstations and the emergence of high speed networks have attracted a lot of interest in parallel and distributed computing (PDC). We envision that PDC environment with supercomputing capability will be available in the near future. Current workstations are capable of delivering tens and hundreds of Megaflops of computing power. A recent report from the IBM European Center for Scientific and Engineering Computing [2] stated that a cluster of 8 RX/6000 Model 560 workstations connected with IBM serial optical channel converter achieved a performance of 0.52 Gigaflops for the Dongarra benchmarks for massively parallel systems. The obtained result outperforms a number of existing parallel computers like a 24 node Intel iPSC/860, 16 node Intel Delta, 256 node nCube2 or a 24 node Alliant CAMPUS/800. Further, workstations are general-purpose, flexible and much more cost-effective, and it has been shown that the average utilization of a cluster of workstations is only around 10% [11]; this unutilized or wasted fraction of the computing power is sizable and, if harnessed, can provide a cost-effective alternative to expensive supercomputing platforms.

Current trend in local area networks is toward higher communication bandwidth as we progress from Ethernet networks that operate at 10 Mbit/sec to higher speed networks such as Fiber Distributed Data Interface (FDDI) networks. Furthermore, it is expected that soon these networks will operate in Gigabit/sec range. However, the application-level transfer rates on existing local area networks remain much lower and it is doubtful that they can keep pace with medium speed. For example, out of the 10 Mbit/sec available at the medium of an Ethernet network, only around 1.2 Mbit/sec bandwidth is available for applications [10].

Consequently, it has been established that current clusters of workstations have the aggregate computing power to provide an environment for high performance distributed computing, while high speed networks, capable of supporting these computing rates, are becoming a standard (e.g., ATM, SONET, HUB-based LAN) [9]. It has also been established that it is not cost-effective to introduce new parallel architectures to deliver the increased computing power. Consequently, we envision that future computing environments need to capitalize on and effectively utilize the existing computing resources. The objective of the presented research is to develop an environment that can harness the computing potential of existing, cost-effective heterogeneous computers and high speed networks.

The organization of the paper is as follows: Section 2 describes an environment for parallel and distributed computing based on hardware and software support that utilizes efficiently the existing heterogeneous computers and the emerging high speed networks. Section 3 analyzes the performance of the communication protocol described in section 2. Section 4 summarizes the paper and provides some concluding remarks.

2 An Environment for Parallel and Distributed Computing

The main objective of the PDC environment is to provide parallel and distributed applications with message passing primitives as well as the host-network interface required to achieve efficient envi-

Figure 2: A configuration of protocol stack with HCP

of protocol stacks as shown in Figure 2. In this subsection, we present briefly an architecture of a host interface that provides the hardware support needed for efficient PDC.

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. In HSM, HIP provides applications with data rate close to that offered by the communication medium. This high speed transfer rate is achieved by (1) using simple communication protocol, HCP, to be discussed in the next subsection (2) decomposing the transmit/receive tasks into several subtasks that can run concurrently on a separate engine and (3) using point-to-point channels that allow all nodes to transmit and receive data concurrently when conflicts are not exist. In NSM, the standard transport protocols can run efficiently on HIP and thus off-load the host from running these protocols. Fig-

Figure 3: Blockdiagram of HIP

of five major subsystems: a Master Processing Unit (MPU), a Transfer Engine Unit (TEU), a crossbar switch, and two Receive/ Transmit units (RTU-1, RTU-2). The architecture of HIP is highly parallel and uses hardware multiplicity and pipeline techniques to achieve high-performance transfer rates. For example, the two RTUs can be configured to transmit and/or receive data over high-speed channels while the TEU is transferring data to/from the host. More details of HIP architecture can be found in [11].

2.2 Software Support for Parallel and Distributed Computing

In this subsection we develop a communication protocol that exploits the support of HIP and provides services needed in PDC. Most existing protocols were designed in the 1970's, when the available communication bandwidths were in the Kb/s range and the existing computing nodes had limited computing power. Since these protocols regarded the communication bandwidth as a scarce resource and the communication medium as inherently unreliable, they were designed to be very general to handle complex failure scenarios, which resulted in complicated protocols implemented as a stack of software layers. The last decade, however, has seen tremendous advances in computing and networking technology. Current networks are highly reliable and can supports high transmission speeds. Further, the computing power of processors has increased while their cost decreased significantly. Consequently, special purpose communication processors like HIP proposed in the previous section can be built to offload hosts from running communication protocols. Furthermore, many services required in parallel and distributed computing can be handled more efficiently by this special interface processor.

We present the design and the implementation of a high speed communication protocol (HCP) that could run on a communication processor such as the HIP. The approach followed in developing HCP 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.

2.2.1 Characterization of Message Passing Primitives for PDC

In oder to identify the HCP services for PDC, we first study the primitives provided by some current parallel/distributed programming tools. The software tools studied include EXPRESS [5], PICL [3], PVM [6], ISIS [1], the iPSC communication library [4] and the CM5 communication library (CMMD) [7]. 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 (CMMD and the iPSC communication library). There is an increased interest in the standardization of message-passing primitives supported by software tools for parallel/distributed computing [8]. 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*, *probe*, or *CMMD_msg_pending* used in EXPRESS, PVM, or CMMD, 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* and *CMMD_send_v/CMMD_receive_v* used in EXPRESS and CMMD, respectively.

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. The regular cases are *scan* (e.g., CMMD's *CMMD_scan*), *concatenation* (e.g., EXPRESS's *exconcat*), *circular shift*, and *all-to-all broadcasting*, while the irregular cases are *gather* and *scatter* (e.g., CMMD's *CMMD_gather_from_nodes*) operation.

Synchronization A parallel / 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, can provide a transparent mechanism to synchronize all the processes. But, 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. Actually, 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.

2.2.2 HCP Message-Passing Primitives

Based on the characterization of message-passing techniques used in parallel/distributed computing

Figure 4: Steps of long message transfer

- **Error and Flow Control:** Sender transmits a frame and then waits for ACK signal from the receiver. When the sender receives a positive ACK (PACK), it sends the next frame; otherwise it retransmits the same frame. Retransmission is repeated a predefined number of times and after that an error signal is raised to the higher layers. The acknowledgment frame serves as a mechanism to achieve flow control between the transmitter and receiver nodes. When the receiver does not have enough buffer space for next frame, it responds with a not-ready indication by setting a flag in ACK frame. If the source receives the not-ready indication from the destination, it stops transmitting frame until it receives ready indication. This simple scheme is attractive because it does not impose

Figure 6: Application-to-application latency

(MPU) selects one of the Receive-Transmit Processor (RTP) to handle the transfer (A). After the Transfer Engine Unit (TEU) is initialized (I) and has started transferring data from the host memory to the buffer (T_1), the RTP sends the Connection Request to the destination node (S_{CR}). On receiving the Connection Confirm, the RTP sends the message data (S_{data}) stored in HNM. The host is notified when the data transfer is complete (N_2). The host then notifies the application (N_3).

At the receiver side, while frames are being received and stored in the (NHM) buffer (R_{data}), the TEU transfers data NHM buffer to the host memory (T_2). When the last frame is received, the RTP sends the disconnect (CC) frame to the sender (S_{DC}). The process R_0 then notifies the host of the message arrival by writing in Common Memory and interrupting the host processor (N_2), which in turns notifies the application (N_3).

The application-to-application latency is indicated in Figure 6 as the time elapsed between the events C and N_3 at the sender and the receiver, respectively. Due to the concurrent operations of the TEU and RTP in the sender such that data transfer from host memory to HIP buffer (T_1) is overlapped with that from the HIP buffer to the network (S_{data}), the latency is minimized. Similarly, the receiving time is also minimized due to the parallel operations of R_{data} and T_2 at the receiver side.

Having analyzed the latency, we consider the transfer rates of long messages. We assume the D-net is lightly-loaded so that no waiting time is consumed at the intermediate nodes when the connection is being established between the source and the destination nodes. The connection establishment will be successful most of the time and the CR frame will not be blocked at intermediate nodes because the CR frame will not be issued unless the required path is available.

We define the application-level data transfer rate $\mathcal{R}$ as the ratio of the data length to be transmitted (l_M) to the total application-to-application transmission time (t_{App}).

$$\mathcal{R} = \frac{l_M}{t_{App}} \tag{1}$$

Based on the discussion in [11], $\mathcal{R}$ can be computed as follows.

$$\mathcal{R} = \frac{l_M}{t_C + t_{N_1} + t_A + t_{setup} + t_{data} + t_{N_2} + t_{N_3}} \quad (2)$$

where t_{setup} denotes the connection setup time, t_{data} represents the time for data transmission and other terms are for the events described earlier.

In Figure 7 and 8, we plot the effective application transmission rate with respect to different message and frame sizes. We consider two channel speeds: 100 Mbit/sec and 1 Gbit/sec. In this analysis we assume the following values for frame fields: 25 byte CR frame, 15 byte length of overhead fields in a data frame, and 15 byte of the ACK frame. Also, we assume that the number of intermediate nodes is 5, the probability of a bit error is 2.5×10^{-10}, the propagation delay between source and destination is $= 0.5$ μsec for average distance of 100 m. Furthermore, we assume each of the following events: $t_C, t_{N_1}, t_{N_2}, t_{N_3}, t_A$ needs around 10 instructions to be processed; i.e. each event can be processed in 1μsec when the performance of HIP processor is 10 MIPS.

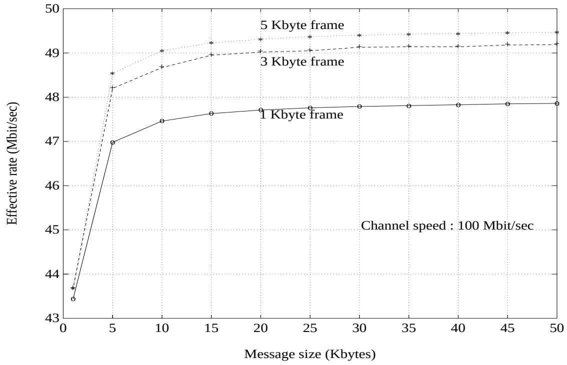

Figure 7: Application-level transfer rate with 100 Mbit/sec channel

4 Conclusion

In this paper, we analyzed the current advances in computing technology, network technology, and software tools for developing parallel and distributed computing applications. We analyzed the primitives, supported by existing parallel and distributed software tools and characterize them into five categories; viz., *point-to-point communication*, *group communication*, *synchronization*, *configuration / control / management*, and *exception handling*. We proposed an environment that capitalizes on the current advances in processing and networking technology and software tools to provide cost-effective parallel and distributed computing. We also presented the design of a communication processor (HIP) that alleviates the host-interface bottleneck and a high speed communication protocol (HCP) that provides the needed bandwidth and services for PDC applications.

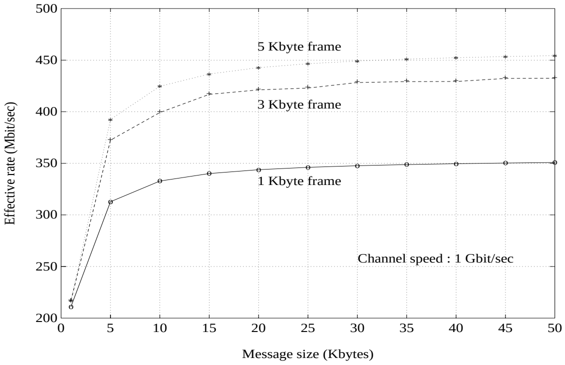

Figure 8: Application-level transfer rate with 1 Gbit/sec channel

References

[1] K. Birman, R. Cooper, T. Joseph, K. Kane, and F. Schmuck. *The ISIS System Manual.*

[2] IBM European Center for Scientific and Engineering Computing. Usenet news item, 1992.

[3] G. A. Geist, M. T. Heath, B. W. Peyton, and P. H. Worley. A user's guide to picl, a portable intrumented communication library. Technical Report Tech. Rep. ORNL/TM-11616, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831, Oct 1991.

[4] Oregon Intel Supercomputer System Devision, Beaverton. ipsc/2 and ipsc/860 user's guide. 1991.

[5] Parasoft Corporation. *Express Reference Manual*, 1988.

[6] V. S. Sunderam. Pvm: A framework for parallel distributed computing. Technical report, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831, 1991.

[7] Massachusetts Thinking Machines Corporation, Cambridge. Cmmd reference manual, version 1.1. 1992.

[8] David W. Walker. Standards for message-passing in a distributed memory environment. *ORNL/TM-12147*, Aug 1992.

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

[10] G. Chesson, " The Protocol Engine Project," *Proceedings of the Summer 1987 USENIX Conference,* pp. 209-215, June 1987.

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