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*

Follow this and additional works at: [https://surface.syr.edu/npac](https://surface.syr.edu/npac)

*Part of the* Computer Sciences Commons

**Recommended Citation**

Hariri, Salim; Park, JongBaek; Parashar, Manish; and Fox, Geoffrey C., "A Communication System for High-Performance Distributed Computing" (1994). *Northeast Parallel Architecture Center*. 35. [https://surface.syr.edu/npac/35](https://surface.syr.edu/npac/35)

This Working Paper is brought to you for free and open access by the College of Engineering and Computer Science at SURFACE. It has been accepted for inclusion in Northeast Parallel Architecture Center by an authorized administrator of SURFACE. For more information, please contact surface@syr.edu.
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 implements 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

* 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 parallel 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.
The organization of the paper is as follows: Section 2 describes an architecture of high-speed local area network (HLAN) and an HPDC environment that utilizes efficiently the existing heterogeneous computers and the emerging high-speed networks. Section 3 identifies a set of message passing primitives by surveying some software tools. Section 4 describes the operation of high-speed communication protocol (HCP). Section 5 analyzes the performance of an application that runs over the proposed HLAN environment. Section 6 summarizes the paper and provides some concluding remarks.

An Environment for HPDC

As network speed increases to Gbit/sec range, communication time between computers is becoming comparable to that between internal components of a computer. We envision that Gigabit LANs will allow computers to interact and collaborate with latency comparable to that between the internal components of a computer. Consequently, future HPDC environment will be equivalent to the current single computer system in terms of the communication latency as shown in Figure 1.

Figure 2 depicts a generalized architecture of a proposed high-speed local area network (HLAN) that aims mainly at providing the required application bandwidth by using a high-speed protocol for HPDC and maintaining at the same time the support of standard protocols. The approach adopted to achieve these
Figure 3: An arc hitecture of HLAN

two goals is based on providing two modes of operation: High-Speed Mode (HSM) and Normal-Speed Mode (NSM). At any given time the system can operate in either or both of these two modes. The HLAN consists of two types of networks: the High-Speed Network (HSNet) and the Normal-Speed Network (NSNet). The HSNet, used during the HSM, consists of two sub-networks, the data network (D-net) and the status/control network (S-net) in a similar manner to the bus structure of a computer system which also can be decomposed into two components (the data bus and status/control bus). The purpose of the S-net is to distribute control and status information about the activities of the computers connected to the D-net and also to support efficient implementation of group communication services. The NSNet is implemented using any standard local area network and is used during the NSM operation. The latency between the components of a computer system shown in Figure 1 will be comparable to those between HPDC components. These networks can be implemented as separate networks or could be logical networks on one physical network.

One possible implementation of the HLAN architecture is shown in Figure 3, where the HSNet employs two ring-type sub-networks. The D-net consists of two counter-rotating channels: while one ring is used for data transmission, the other ring is used for acknowledgements. The D-net is a point-to-point network where each channel segment between nodes works independently, not in a shared manner as in token-ring networks. However, the S-net is a broadcast network based on token-ring scheme. In this paper, we study the design of a host-interface processor (HIP) and a high-speed communication protocol to support such an environment.

Another implementation of the HLAN architecture is based on the ATM network as shown in Figure 4. Instead of having different physical sub-networks for the HSNet and NSNet, the ATM-based HLAN implements them on two logical sub-networks sharing one physical ATM channel; by allocating more network bandwidth to the HSM, both modes of traffic can be multiplexed over the channels of the ATM-based HLAN. The traffic associated with the HSM is carried out through the solid lines (which represents a large percentage of the aggregate bandwidth of the physical channel), while the NSM traffic is delivered through the dotted lines.

Figure 5 depicts an HPDC environment that provides applications with message passing primitives to...
Figure 4: An ATM-based HLAN achieves consistent environment over high-speed networks. The software portion of this environment consists of a high-speed communication protocol (HCP) and a HCP run-time system. The HCP run-time system is an interface between a parallel and distributed programming tool and the HCP services running on an interface processor. In a distributed programming environment, software tools such as EXPRESS or PVM provide a communication library for message passing. The current implementations of these tools utilize low-level communication programming interfaces (e.g., BSD socket library) that are supported by a standard transport protocol (e.g., TCP/IP). Because these interfaces involve a large number of system calls, data copying and memory management, they cannot provide the high-bandwidth and low-latency communication needed for HPDC. To solve the problems above, HCP provides all the services (data transfer, synchronization and control) needed for efficient parallel and distributed computing. Furthermore, these services run on a host interface processor (HIP) and therefore offload the host. In next sections, we present hardware and software support to build such HPDC environment.
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/exvrecv used in EXPRESS.

7
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, sync, 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, and 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 exclusive access of resources shared between processors. The hcp_barrier primitive enables a specified number of processors to synchronize at a logical barrier before 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 logical 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 log 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),
Table 1: A characterization of message-passing primitives for parallel and distributed computing

- **Transfer Engine Unit (TEU)**, a crossbar switch, and two Receive/Transmit units (RTU1, RTU2).

  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 transfers data to/from the host. More detailed description for HIP can be found in [5, 1].

---

**Point-to-Point Data Transfer over the D-net**

HCP is the protocol for the HSNet portion of HLAN. It supports two types of communication: point-to-point communication over the D-net and multicasting communication over the S-net. In this subsection, we describe the operation of the point-to-point data transfer.

Each node participating in a computation over the D-net is in one of the following modes during its operation: Idle (ID), Receive-only (RO), Transmit-only (TO), Receive and transmit (RT), Receive-and-Receive (RR), Transmit-and-Transmit (TT) or Bypass (BP) mode. Initial mode is ID. In BP mode, a node is just isolated from the network and all the incoming data is forwarded to the next node with minimum delay. Figure 7(a) shows all possible mode transitions for a node. Figure 7(b) demonstrates a case in which node 0 is transmitting data to node 2 and 6, node 5 is receiving data from node 4 and 6. Note that there are 4 circuit connections established at the same time. The operation mode of each node is periodically broadcasted over the S-net.
Table 2: HCP services

We distinguish between two transfer schemes depending on the message size: long message transfer and short message transfer. A message with length of less than a data frame size is designated as a short message and otherwise it is regarded as a long one. Each long message is transferred as a sequence of multiple data frames. The size of a data frame is determined to be as large as possible because larger frames perform better as will be shown later. However, the maximum frame size should be within the limit where clock skewing does not lead to a synchronization problem at the receiver.

For long messages, data transmission is performed in two phases: connection setup and data transfer. Figure 8 shows all the steps involved in long message transfer; establishing a connection, receiving a confirmation of a successful connection, transferring the data frames, and finally disconnecting the connection, respectively.

Connection setup phase: The source node initiates data transfer by sending a connection request (CR) frame to the destination node when it determines that all intermediate nodes, if any, are in ID mode (see Figure 8(a)). This scheme will reduce probability for the CR to be blocked by intermediate nodes. If an intermediate node in the path remains in ID mode until the CR frame arrives at the node, it is changed to BP mode (this is highly probable because the CR will not be sent unless all intermediate nodes are in the ID mode); otherwise the CR frame waits at the intermediate node which has changed its mode while the CR frame was traveling from the source node to that node. The CR frame will be forwarded when the channel becomes available. This process is repeated until the CR frame reaches the destination node. The CR frame carries the information of message length and data frame size so that the receiver can identify
Once the CR reaches the destination node, it responds with another control frame, connect confirm (CC), which will be sent to the source in the opposite direction using the other ring (Figure 8(b)). No channel arbitration or waiting is needed for sending CC frame to the source since circuit is already established.

**Data transfer phase:** Once connection is established, data is transferred as multiples of frames (Figure 8(c)). Simple scheme for error and acknowledgment control is used as described later. Since the receiver knows the number of frames to be transferred, it automatically sends the disconnect (DC) signal when it receives the last frame with no error; i.e., the last frame is acknowledged with DC frame if correctly received as shown in Figure 8(d). Intermediate nodes should be able to detect the DC frame while they are in BP mode and then switch their state to the appropriate modes.

**Short Message Transfer**
In this case, the data is transmitted with the CR frame. Consequently, the connection setup and data transfer phases of long message transfer are combined into one step. When it passes intermediate node, the node changes to bypass mode until a DC frame is received from the destination node. Once the frame reaches the destination, it responds with DC signal when the data received is in no error; the frame is treated as a short message transfer.
Figure 7: Operation modes

Figure 8: Steps of long message transfer

as the last frame of a long message. This scheme reduces the connection setup overhead by eliminating the connection confirm (CC) step in long message transfer.

Error and Flow Control

The sender transmits a frame and then waits for ACK signal back from receiver. When the sender receives a positive ACK (PA CK), 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 acknowledgement frame is used to achieve flow control between the transmitter and receiver nodes. When receiver does not have enough buffer space for next frame, it responds with a not-ready indication by setting again ACK frame. If the source receives the not-ready indication from the destination, it stops transmitting frame until it receives a ready indication. This simple scheme is attractive because it does not impose any limit of the transmission rate that could be in Gigabit or Terabit range; it is doubtful that the current error and flow control methods used in existing standard protocols can cope with such high transmission rate.\cite{1/4/3}

Owing to the simple error/flow control scheme, HCP can cope and handle with mediums operating in Gigabit or even Terabit range; in existing standard protocols, at high transmission rates, the network interface processor would have to process an incoming packet within a very short time interval. For example, let us suppose that /1/0/0/0 instructions are needed for processing /1 Kb yte packets /\cite{1/4/1/3}/. For a network operating in /1/0/0 Mbit/sec and /1 Gbit/sec/, the interface would have to process the incoming packet in /8/0 sec/, and /8 sec/, respectively. Consequently, the network interface processor speed must have at least /1/2/./5 MIPS and /1/2/5 MIPS capability for maximum throughput. It is clear from this simple analysis that the existing protocols\cite{1/3/}...
Figure 9: Frame format for D/-net

Figure 10: Frame formats for S/-net

cannot be scaled up to Gigabit or Terabit rates and new protocols such as HCP must be developed.

Frame Formats

In Figure 9, we show four types of frames which are used in D/-net during HSM: CR frame with short data, CR frame with long data, Data frame, and ACK frame. The preamble field (P A) is used to achieve synchronization of receiver clock with sender’s. The StartingDelimiter (SD) field and EndingDelimiter (ED) field denote the start and end of the frame, establishing unique frame boundaries. The type field differentiates the four kinds of frames. The Source (SRC) and Destination (DST) fields in CR frame indicate the network address of source and destination nodes and the length field represents the size of data to be transmitted in bytes. Due to the length field and frame size field in CR frame for long data, the receiver can identify how many frames will arrive from the source node, i.e., number of frames to receive, \( n = \frac{d_{\text{length}}}{\text{frame size}} \).

The status field in ACK frame distinguishes acknowledgments of connection confirm (CC) and disconnect (DC) as well as positive (PACK) and negative (NA CK) acknowledgment of data frames. The RDY field in ACK frame denotes the availability of the buffer space at the receiver. The checksum field (CHK) is based on a cyclic redundancy code to detect errors in received frames.

Multicasting over the S/-net

Group communication primitives such as multicasting and broadcasting are important for HPDC. HCP protocol utilizes the broadcast capability of the S/-net to efficiently implement the group communication primitives. The S/-net access protocol is an adaptation of a standard token ring protocol. In this protocol the token not only controls the access to the S/-net, but also propagates the status/control information associated with each node. Whenever a node receives the token, it writes its status in the designated field of the token frame as shown in Figure 10. If this station has data to send to a group of processes (multicasting/broadcasting), it puts the data on the S/-net and then releases the token so that it can be picked up by the next node. Figure 10 shows frame formats for both token and data frames. The GA field in the data frame represents the group address for multicasting; any node whose group address corresponds to...
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:

```plaintext
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 * n - 1 /* for the subsequent blocks */
    if my_num = i mod N
```
Figure 1: An example of LU factorization:

\[ N = 4 \text{ and } n = 2 \]

- Receive the previous factorization result \( A \);
- Update block \( i \) using \( A \);
- Factorize block \( i \) and obtain result \( B \);
- Broadcast \( B \);
- Update the remaining blocks using \( A \);
- Update the remaining blocks using \( B \);

- If the number of remaining blocks > 0
  - Receive the result \( B \);
  - Update the remaining blocks;

The operations in the algorithm, Factorize and Update, are further analyzed in Figure 1, where an intermediate step (step 3) of the case of \( N = 4 \text{ and } n = 2 \) is shown. The computations involved are:

1. Factorization of the block \( C \) to obtain \( L_1, L_2 \), and \( U_1 \), which consists of (see Figure 1(b))
2. Computation of column of \( u_j \):
   \[ u_j = (L_j) ?^1 \]
3. Update of column of \( l_j \):
   \[ l_j = l_j ?^A_j \]
4. Pivot (piv) selection in \( l_j \) and consequent row interchanges
5. Scaling of column :
   \[ l_j = l_j = piv \]
6. Computation of \( U_2 \):
   \[ U_2 = (L_1) ?^1 \]
7. Update of the bottom right blocks:
   \[ B = B ?^L_2 \ U_2 \]

Note that step 2) and 3) are computed in parallel (each computer updates the portion allocated to it).

Performance Analysis

Let us assume that the size of the matrix is \( m \times m \) and the same number of blocks are allocated to each computer. Some variables which will be used in the analysis are defined as follows:

1. 
Figure 1/2: Timing diagram for LU decomposition

- $N$ = number of computers
- $b$ = block size (number of columns)
- $n$ = number of iterations or number of blocks allocated to each computer
- $N/n$ = total number of steps

Since the results of each factorized block is used by all the other processors, the HCP broadcast primitives will be used to send the factorized block to all computers.

- $i$ = 1, 2
- $F(i)$ = factorization time at step $i$
- $U(i; j)$ = update time for $j$ remaining blocks at step $i$

$U(i; 0) = 0$

$F(i)$ = broadcast time at step $i$

From Figure 1/2, we can estimate the computation time $T_{LU}$ as follows:

$$T_{LU} = F(1) + \sum_{i=2}^{N/n} [comm(i) + \max\{A; B; C\}]$$

Where

$A = U(i; 0) + U(i; n - 1) + b$

$B = U(i; 1) + b$

$C = U(i; 1) + F(i)$
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:

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

$$Speedup_{LU} = \frac{(2/3) \cdot m^3 \cdot (1/MFLOPS_{computer})}{T_{LU}}$$  \hspace{1cm} (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)](image1)

![Figure 14: Effective MFLOPS for LU decomposition (40 MFLOPS computers)](image2)
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


