1993

Architectural Support For High-Performance Distributed Computing

JongBaek Park
Syracuse University

Salim Hariri
Syracuse University

Recommended Citation

Park, JongBaek and Hariri, Salim, "Architectural Support For High-Performance Distributed Computing" (1993). *Electrical Engineering and Computer Science*. 111.
https://surface.syr.edu/eecs/111

ARCHITECTURAL SUPPORT FOR HIGH-PERFORMANCE DISTRIBUTED COMPUTING

JongBaek Park and Salim Hariri

Electrical and Computer Engineering
Syracuse University
Syracuse, NY 13244

Abstract

The emergence of high speed networks and the proliferation of high performance workstations have attracted a lot of interest in workstation-based distributed computing. 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 that can operate in Gbit/sec range. Also, current workstations are capable of delivering tens and hundreds of Megaflops of computing power. By using a cluster of such high-performance workstations and the high-speed networks, a high-performance distributed computing environment could be built in cost-effective manner as an alternative of supercomputing platform.

However, in current local area networks, the bandwidths achievable at the application level are often an order of magnitude lower than that provided at the network medium [3, 7]. It is therefore not sufficient to have even a Gigabit data link if user applications could only use a small portion of that bandwidth.

In this paper, we present a software and hardware support to transform a local area network of workstations into a high-performance distributed computing environment. We present a Host Interface Processor (HIP) and a communication protocol (HCP) in order to improve the application-level transfer rates. We also analyze the performance of a distributed application when it runs on the computers of the HIP-based local networks and compare it with the performance of a single computer execution.

1 Introduction

The emergence of high speed networks and the proliferation of high performance workstations have attracted a lot of interest in workstation-based distributed computing. 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 FDDI networks. Furthermore, it is expected that soon these networks will operate in Gbit/sec range. Also, current workstations are capable of delivering tens and hundreds of Megaflops of computing power; for example, a cluster of 1024 DEC alpha workstations would provide 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 [1]. Hence, the aggregate computing power of a group of general purpose workstations can be comparable to that of supercomputers. Further, workstations are general-purpose, flexible and much more cost-effective. Furthermore, it has been shown that the average utilization of a cluster of workstations is only around 10% [4]; 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. Consequently, we project that current clusters of workstations have the aggregate computing power to provide an supercomputing environment with the support of high speed networks.

However, a number of issues have to be resolved in order to utilize the full potential of workstation-based supercomputing environments. The primary barrier is the limited communication bandwidth available at the application level. In current local area networks (LAN), the bandwidths achievable at the application level are often an order of magnitude lower than that provided at the network medium [3, 7]. For example, out of the physical bandwidth of 10 Mbit/sec available at the medium level of the Ethernet, only about 1.2 Mbit/sec is available to the application [3]; it is therefore not sufficient to have even a Gigabit data link if user applications could only use a small portion of that bandwidth. This degradation in performance occurs because of two main reasons: (1)Host-to-network interface characterized by its excessive overhead of processor cycles and system bus capacity, heavy usage of timers, interrupts, and memory read/writes; and (2)the standard protocols implemented as a stack of software layers which consume most of the medium capacity and provide very little bandwidth to the application.

Recently, there has been an increased interest to improve the transfer rate at the application-level by introducing new high-speed transport protocols. The general concepts for high-speed protocols can be characterized as follows [6]: (1)design philosophies; (2)architecture philosophies; and (3)implementation philosophies. Although these projects have resulted in reducing communication overhead and improving the application transfer rates, they still support local network protocols that allow only one computer to transmit at a time.

In this paper, we present the architecture of a High-speed Communication Protocol (HCP) and a Host Interface Processor (HIP) that aim mainly at providing the efficient application bandwidth, and maintaining at the same time the support for standard protocols. HCP is characterized as its simple communication scheme to provide low latency and high bandwidth, and concurrent communication capability to allow multiple hosts to communicate over local networks at the same time. HIP is a multiproces-

checking is done by a library routine (C), another routine writes a send request in the Common Memory of HIP (N_1) and then interrupts the Master Processor. The send request includes the address of the destination node and a pointer to the message and its size. The Master Processing Unit (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 7 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. The same method can be applied to analyze short message transfer rates. 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; the S-net provides the status information required to determine whether the required path is available or not.

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}).

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

This transmission time can be approximated as

$$t_{App} \simeq t_C + t_{N_1} + t_A + t_{setup} + t_{data} + t_{N_2} + t_{N_3} \tag{2}$$

where we assume that by the time the ACK frame of the last frame is received and the connection is released, the TEU at the receiver has also completed the transfer of the data to the host memory.

The connection setup time t_{setup} consists of the time for CR frame preparation (t_{prep}), transmission time of the CR frame (t_{CR}), processing time for the CR at each intermediate nodes ($t_{proc,CR}$), transmission time of ACK (t_{ACK}, i.e., connection confirm in this case), and the round trip propagation delay ($2 \cdot t_p$) for the CR and ACK frame.

$$t_{setup} = t_{prep} + t_{CR} + k \cdot t_{proc,CR} + t_{ACK} + 2 \cdot t_p \tag{3}$$

where k denotes the number of intermediate nodes. Once the connection is established, data is transmitted as multiples of data frames. Since there is only one outstanding frame to be acknowledged, we can estimate the average time to transmit a data frame as follows. Let t_T be the time to send a data frame and receive an acknowledgment which is either a PACK, if correctly delivered, or a NACK, in erroneous transmission. Then,

$$t_T = t_{frame} + t_{ACK} + 2 \cdot t_p + t_{proc,src} + t_{proc,dest} \tag{4}$$

which includes the data frame transmission time t_{frame}, the time for ACK frame transmission t_{ACK}, the round trip propagation delay, and the processing time for the data frame at the source and the destination nodes $t_{proc,src}$,$t_{proc,dest}$, respectively. We further break the t_{frame} into two parts since we are interested in application-to-application data transmission time: the time to transmit only the data field portion in each data frame t_m and the overhead field transmission time for the rest of the data frame t_h. Therefore,

$$t_T = t_m + t_h + t_{ACK} + 2 \cdot t_p + t_{proc,src} + t_{proc,dest} \tag{5}$$

The transmission time of a frame can be determined probabilistically assuming that each frame fails independently. Let P be the probability of a frame being received in error. Then, using the geometric mean, the expected transmission time for a frame can be evaluated as

$$t_a = \frac{t_T}{1-P} \tag{6}$$

Hence, the total transfer time of a message data t_{data} can be expressed as

$$t_{data} = n_f \cdot t_a \tag{7}$$

where n_f is the number of data frames, i.e., $n_f = \lceil l_M / l_m \rceil$, where l_m is the length of data field, in bits, of a data frame. Note that the connection release time is included in the t_{data} since the acknowledgment of the last frame implies connection release from receiver.

Consequently, from (1), (2), (3), (5), (6) and (7), we obtain

$$\begin{aligned}\mathcal{R} \;=\; & \frac{l_M}{\begin{aligned} & t_C + t_{N_1} + t_A + t_{prep} + t_{CR} + t_{ACK} + k \cdot t_{proc,CR} \\ & +2 \cdot t_p + (n_f/(1-P)) \cdot \{t_m + t_h + 2 \cdot t_p + t_{ACK} \\ & +t_{proc,src} + t_{proc,dest}\} + t_{N_2} + t_{N_3}\end{aligned}}\end{aligned} \tag{8}$$

In Figure 8 and 9, 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: length of the CR frame $l_{CR} = 25$ bytes, length of overhead fields in a data frame $l_h = 15$ bytes, length of the ACK frame $l_{ACK} = 15$ bytes. Also, we assume that the number of intermediate nodes is k= 5, the probability of a bit error is $p = 2.5 \times 10^{-10}$, the propagation delay between source and destination is t_p = 0.5 μsec for average distance of 100 m, and the $t_{CR,proc}$ is 1 μsec. Furthermore, we assume each of the following events: $t_C, t_{N_1}, t_{N_2}, t_{N_3}, t_A, t_{prep}$ needs around 10 instructions to be processed; i.e. each event can be processed in 1μsec if the processor speed is 10 MIPS. Also, we assume that each of the events ($t_{proc,src}$ and $t_{proc,dst}$) takes around 20 instructions and therefore can be processed in 2 μsec.

Note that in a conventional token ring network, since communication is sequential, the total number of communications is $N \cdot \log_2 N$; the communication overhead is reduced, for this application, in the order of $\log_2 N$. Since we are transmitting K results in each message, the communication time for each message t_{comm} is computed as

$$t_{comm} = \frac{K \cdot l_{data}}{\mathcal{R}}$$

where l_{data} is the length of an intermediate results (e.g., number of bits representing a complex number) and $\mathcal{R}$ is the application-to-application transfer rates evaluated in the previous section. Therefore, the total communication time $T_{comm,HLAN}$ is

$$T_{comm,HLAN} = (N-1) \cdot K \cdot l_{data} \cdot \frac{1}{\mathcal{R}} \tag{10}$$

Because the computations at each node are executed in parallel, the total computation time is given by

$$T_{comp,HLAN} = t_{op} \cdot \log_2 M \tag{11}$$

Therefore, combining (9), (10) and (11, we obtain

$$Speedup = \frac{N \cdot t_{op} \cdot (\log_2 N + 1)}{(N-1) \cdot K \cdot l_{data}/\mathcal{R} + t_{op} \cdot (\log_2 N + 1)} \tag{12}$$

Figure 11 shows the speedup gain and the effective MFLOPS with respect to different number of computers. It is clear from this figure the potential increase in the speedup and the computing power when HLAN operates at high-speed transmission rate. For example, for 1 Gigabit HLAN, the total MFLOPS provided to the FFT application could reach 250 MFLOPS when 60 computers with speed of 10 MFLOPS are used. However, for 100 Mbit HLAN, this rate will be reduced to 60 MFLOPS for 60 computers. It is important to notice that these rates are the rates provided to the applications and are much higher than those provided by existing standard protocols [3]. This simple analysis demonstrates the potential performance gain that can be achieved when the distributed computing is supported with communication software and hardware that provides application bandwidth comparable to that offered by the medium.

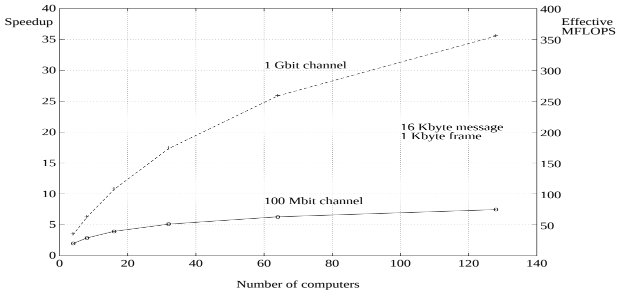

Figure 11: Speedup and effective MFLOPS with respect to single computer execution

6 Conclusion

In this paper, we presented an architecture for a high-speed communication protocol (HCP) that provides application with bandwidth comparable to that offered by transmission lines. This protocol can transform a local network of heterogeneous computers into a high-performance distributed computing environment suitable for compute-intensive applications. HIP-based LAN provides the architectural support needed to improve the user-level transfer rates, supports both standard and nonstandard fast transport protocol, and efficient distributed processing over the network and better utilizations of idle computing power available across the network. HLAN operates in two modes of operation: Normal-Speed Mode (NSM) where a standard transport protocol is used to transmit and/or receive data over a channel allocated to this mode; and High-Speed Mode (HSM) where processes can bypass the standard transport layers and access directly the HIP software layer to achieve application transfer rates comparable to the medium speed.

References

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

[2] 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.

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

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

[5] B. Beach, " UltraNet: An Architecture for Gigabit Networking," *the 15th Conference on Local Computer Networks*, pp. 232-248, October 1990.

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

[7] 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, 1987.

[8] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Architecture*, McGraw-Hill, 1984.

[9] R. Jain," Performance Analysis of FDDI Token Ring Networks: Effect of Parameters and Guidelines for Setting TTRT," *IEEE LTS*, May 1991.

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