Clemson University
TigerPrints

Publications | School of Computing

4-1999

Evaluation of Real-Time Fiber Communications for Parallel Collective Operations

Amy Apon
Clemson University, aapon@clemson.edu

Parvathi Rajagopal
Vanderbilt University

Follow this and additional works at: https://tigerprints.clemson.edu/computing_pubs

 Part of the Computer Sciences Commons

Recommended Citation

Apon, Amy and Rajagopal, Parvathi, "Evaluation of Real-Time Fiber Communications for Parallel Collective Operations" (1999). *Publications* . 23.
https://tigerprints.clemson.edu/computing_pubs/23

This is brought to you for free and open access by the School of Computing at TigerPrints. It has been accepted for inclusion in Publications by an authorized administrator of TigerPrints. For more information, please contact kokeefe@clemson.edu.

Evaluation of Real-Time Fiber Communications for Parallel Collective Operations*

Parvathi Rajagopal[1] and Amy W. Apon[2]

[1] Vanderbilt University, Nashville, TN 37235 parvathi@dg-rtp.dg.com
[2] University of Arkansas, Fayetteville, AR 72701 aapon@comp.uark.edu

Abstract. Real-Time Fiber Communications (RTFC) is a gigabit speed network that has been designed for damage tolerant local area networks. In addition to its damage tolerant characteristics, it has several features that make it attractive as a possible interconnection technology for parallel applications in a cluster of workstations. These characteristics include support for broadcast and multicast messaging, memory cache in the network interface card, and support for very fine grain writes to the network cache. Broadcast data is captured in network cache of all workstations in the network providing a distributed shared memory capability. In this paper, RTFC is introduced. The performance of standard MPI collective communications using TCP protocols over RTFC are evaluated and compared experimentally with that of Fast Ethernet. It is found that the MPI message passing libraries over traditional TCP protocols over RTFC perform well with respect to Fast Ethernet. Also, a new approach that uses direct network cache movement of buffers for collective operations is evaluated. It is found that execution time for parallel collective communications may be improved via effective use of network cache.

1 Introduction

Real-Time Fiber Communications (RTFC) is a unique local and campus area network with damage tolerant characteristics necessary for high availability in mission critical applications. The RTFC standard [5] has been in development over a period of several years and first generation products are available through two independent vendors. RTFC arose from the requirements of military surface ship combat systems. Existing communication systems, while fast enough, did not have the necessary fault and damage tolerance characteristics. When constructed in a damage-tolerant configuration, RTFC will automatically reconfigure itself via a process known as rostering whenever damage to a node, link, or hub occurs.

In addition to its damage-tolerant features, RTFC is a ring that uses a connectionless data-link layer that has minimum overhead [9]. Unlike switched technologies such as Myrinet and ATM that may perform multicast and broadcast

* This work was supported in part by NSF Research Instrumentation Grant #9617384 and by equipment support from Belobox Systems, Inc., Irvine, California.

communications by sending unicast messages to all destinations, RTFC directly supports the multicast and broadcast communications that are important in parallel applications. Therefore, it seems appropriate to explore the advantages that RTFC may have in parallel collective communications (i.e., those that are variants of broadcast and multicast messaging).

The purpose of this paper is to introduce Real-Time Fiber Communications and to evaluate its applicability for parallel collective operations. The evaluation is based on measurements and experimentation on a cluster of workstations that is interconnected via RTFC. The paper is organized as follows.

- Section 2 introduces Real-Time Fiber Communications,
- Section 3 describes the experimental workstation and network platform,
- Section 4 describes the software platform for collective communications,
- Section 5 gives the measurement results, and
- Section 6 gives conclusions and a description of related and future work.

2 Overview of Real-Time Fiber Communications

The basic network topology of RTFC is a set of nodes that are connected via a hub or switch in a simple star configuration. The nodes form a logical ring, and each node has an assigned number in increasing order around the ring. One physical star forms a single non-redundant fault-tolerant segment. The maximum number of nodes in a segment is 254. The distance from a node to a hub or switch is normally up to 300 meters for 62.5 micro fiber-optic media. For applications that do not require high levels of fault tolerance, it is also possible to configure RTFC as a simple physical ring using only the nodes and the network interface cards in the nodes. At the physical layer RTFC is fully compliant with FC-0 and FC-1 of the Fibre Channel FC-PH specification.

The RTFC ring is a variant of a register insertion ring [1], and guarantees real-time, in-order, reliable delivery of data. During normal operation, packets are removed from the ring by the sending node (i.e., a source removal policy is used). While this limits the maximum throughput on the ring to the bandwidth of the links only, this policy has certain advantages in that sending nodes are always aware of traffic on the entire ring. Since each node can see all of the ring traffic, it is possible for nodes to monitor their individual contribution to ring traffic to ensure that the ring buffers do not fill, and that ring tour time is bounded. The unique purge node in the segment removes any orphan packets that traverse the ring more than one time.

Bridge nodes may be used to connect segments, and redundant bridge nodes may be used to connect segments to meet the damage-tolerance requirements of a multi-segmented network. For applications that require high levels of fault tolerance, it is possible to construct redundant copies of the ring. By adding channels to the network interface cards at each node and adding additional hubs or switches, up to four physically identical rings may be constructed with the same set of nodes. In this manner, RTFC can be configured to be dual, triple,

or quad-redundant for fault tolerance. The communication path from one node to the next higher-numbered node in the ring goes through a single hub. While other hubs receive a copy of all packets and forward them, the receiving node only listens to one hub, usually the one with the lowest number if no damage has been sustained. If a node, hub, or communication link fails in an RTFC segment, then a process known as rostering determines the new logical ring. This ability to reconfigure the route between working nodes (i.e., the logical ring), in real-time, after sustaining wide-scale damage, is a distinguishing characteristic of RTFC.

RTFC supports traditional unicast, multicast, and broadcast messaging between nodes. In addition, RTFC supports the concept of network cache. Network cache is memory in the network interface card that is mapped to the user address space and is accessible to the user either via direct memory access (DMA) or via copy by the CPU. One use of network cache is to provide an efficient hardware implementation of distributed shared memory across the nodes in a segment. Each network interface card can support up to 256MB of network cache. Some portion of network cache is reserved for system use, depending on the version of system software running, but the remaining portion of network cache can be read or written by user processes running on any workstation in the network.

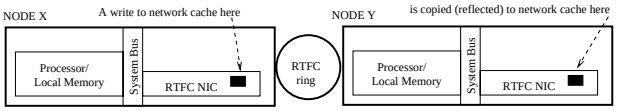

Fig. 1. Operation of the RTFC network cache

Broadcast data written to network cache of any node appears in the network cache of all nodes on that segment within one ring-tour, as illustrated in Figure 1. The ring-tour time is the time that it takes a micropacket to travel from the source around the ring one time. The total hardware latency to travel from the source around the ring one time can be calculated to be less than one millisecond on the largest ring at maximum distance [5].

The data link layer in RTFC is defined by the RTFC standard and replaces the FC-2 layer defined in Fibre Channel. Data at the data link layer is transferred primarily using two sizes of hardware-generated micropackets. For signals and small messages, a fixed-size, 12-byte (96 bit), micropacket is used [9].

The data rate of RTFC can be calculated. RTFC sends at the standard Fibre Channel rate of approximately 1 Gbps. Fibre Channel uses 8B/10B data encoding, giving a user data rate of 1 Gbps * 80 % = 800 Mbps. For each micropacket the percentage of data bits is (32 data bits)/(96 bits/micropacket + 32 bits/idle) = 25%. Therefore, the user data rate is approximately 200 Mbps. For longer messages, a high-payload micropacket longer than 12 bytes is supported in hardware that obtains up to 80% efficiency of the theoretical fiber optic bandwidth. All testing was performed with first generation hardware that only supports 96 bit micropackets.

3 Experimental Workstation and Network Platform

The workstations used in this study consists of three dual-processor VME bus-based workstations, connected by a 3Com SuperStack II Hub 100 TX Fast Ethernet hub for 100Mbps tests and a VMEbus RTFC gigabit FiberHub. Each workstation is identified by an Ethernet network IP address and host name, and similarly by an RTFC network IP address and host name.

A simple block diagram of the workstation hardware is shown in Figure 2. Each of the three workstations has a Motorola MVME3600 VME processor module, as outlined by the dashed line in the figure. The MVME3600 consists of a processor/memory module and a base module, assembled together to form a VMEbus board-level computer. The base module and processor/memory modules are both 6U VME boards. The combination occupies two slots in a VME rack. The processor/memory modules consists of two 200MHz PowerPC 604 processors and 64MB of memory. The processor modules have a 64-bit PCI connection to allow mating with the MVME3600 series of baseboards. The base I/O module consists of a digital 2114X 10/100 Ethernet/Fast Ethernet interface and video, I/O, keyboard, and mouse connectors. The MVME761 transition module (not shown in Figure 2) provides a connector access to 10/100 BaseT port. A P2 adapter (not shown in Figure 2) routes the Ethernet signals to the MVME761 transition module. Each workstation is connected to the RTFC network via a V01PAK1-01A FiberLAN SBC network interface, each with 1MB of SRAM network cache memory.

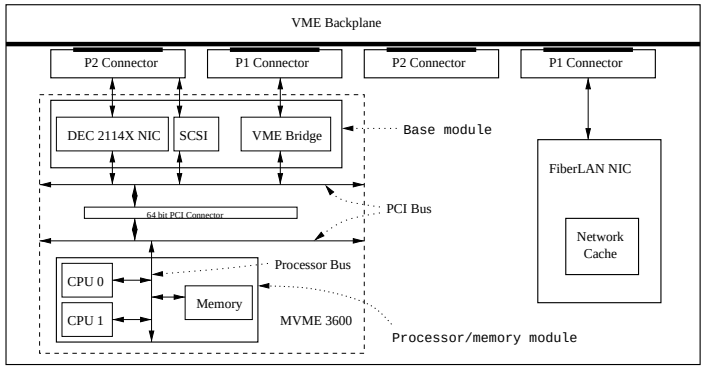

Fig. 2. Simple block diagram of the workstation architecture

System software on the workstations is the IBM AIX 4.1.5 operating system. The MPICH implementation of the MPI (Message Passing Interface) is chosen as the application platform [4]. MPICH network communication that is based on TCP/IP can be made to run over Fast Ethernet or over RTFC, depending on the host names that are specified in the proc group configuration file. The MPICH implementation of MPI provides unicast (point-to-point)

communications, broadcast, and multicast (collective) communications. MPICH collective communications are implemented using point-to-point, which are implemented through message passing libraries built for the specific hardware. The performance of the collective operations in MPICH is enhanced by the use of asynchronous (non-blocking) sends and receives where possible, and by using a tree-structured communication model where possible.

In addition to TCP/IP, messaging is provided over RTFC via the Advanced MultiProcessor/Multicomputer Distributed Computing (AmpDC) environment [2], a product of Belobox Systems, Inc. However, an implementation of AmpDC middleware for RTFC was not yet available for AIX. At the time of this study, a partial implementation of some RTFC primitive components for AIX were available, including a function to write data directly to network cache memory from local workstation memory using DMA or CPU access, and a function to read data directly from network cache to local workstation memory. Also available were two functions that return a pointer to the first and last available long word location of user-managed network cache, respectively.

Standard benchmarking techniques were used throughout this study. All runs were executed for at least one hundred trials. The timings obtained were cleared of outliers by eliminating the bottom and top ten percent of readings. The network was isolated in that other traffic was eliminated and no other applications were running on the workstations at the time. Each call was executed a few times before the actual timed call was executed in order to eliminate time taken for setting up connections, loading of the cache, and other housekeeping functions. Measurements were made on both RTFC and Fast Ethernet.

4 Software Platform for Collective Communications

An experimental software platform was developed for evaluating the performance of MPI collective communications by using the network cache architecture [8]. Collective calls for broadcast, scatter, gather, allgather, alltoall were designed to work similarly to their corresponding MPI calls. These calls were implemented on the experimental platform by using the basic RTFC read and write calls.

In order to allow processes to not over-write each other's data, network cache was partitioned among the processes [6]. Each process writes to its area of network cache. Each process may read from one or more areas of network cache, depending on the specific collective operation. An initialize function finds the first and last location on network cache that is available to the user applications, and partitions the available memory equally among the processes. The assumption is made that at any given time there is only one parallel application (i.e., MPI program) executing, and that this application has access to all of network cache that is available to user programs. If there is a need to execute more than one parallel program at a time then an external scheduler should be used that will be responsible for the additional partitioning of network cache that would be required.

Collective calls are considered complete when the area of network cache used by the process can be reused. Since the experimental platform does not yet support interrupts at the receiving node via the RTFC primitives available at the time of these tests, there is a difficulty in how to signal a receiving node that a message has arrived. In order to demonstrate proof-of-concept, MPI messaging over Fast Ethernet is used in the prototype code to notify a set of nodes of the start of each collective operation and to synchronize nodes between collective operations.

In MPI, processes involved in a collective call can return as soon as their participation in the routine is completed. Completion means that the buffer (i.e., the area of network cache) used by the process is free to be used by other processes. The method for signaling completion of each of the calls varies depending on the operations being performed. In general, the sending processes write to network cache and inform all other processes where to find the data in network cache, the receiving processes read the data from the network cache, and an MPI synchronization call ensures that the buffer is safe before it is re-used.

For example, in case of the broadcast, the root writes the contents of the send buffer to its area in network cache and sends the starting address to all other processes in the group. The other processes receive the address, read the information from network cache and write it to their local memory. Once the processes write the message to local memory, the send a completed message to the root which exits upon receing the complete message from all receivers.

As another example, in case of allgather, each process sends the start address to all other processes. That is, the address is allgathered to other processes. Once a process reads the contents of network cache it sends a completed message to the sending process, which then exits upon receiving the completed message. Figure 3 illustrates the flow of execution for allgather. Figure 4 illustrates the partitioning of network cache and the use of local memory for allgather. Similar algorithms are implemented for scatter, gather, and alltoall messaging.

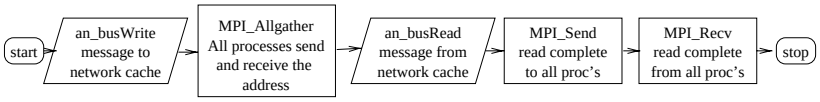

Fig. 3. Flowchart for the allgather operation

5 Measurement Results for Collective Communication

Performance measurements were conducted for the experimental code running over RTFC, and also for MPI over TCP/IP on both RTFC and Fast Ethernet. The time from the start of the collective call until the completion of the last process is the metric used in this study [7]. Measurements were performed for broadcast, alltoall, allgather, scatter and gather, for two and three processes.

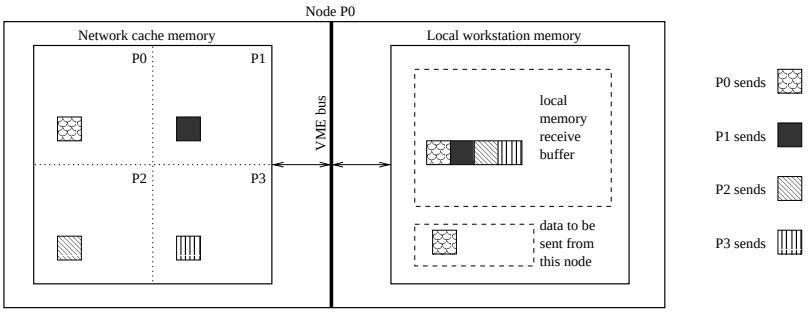

Fig. 4. Partitioning of network cache for the allgather operation (shown at node P0)

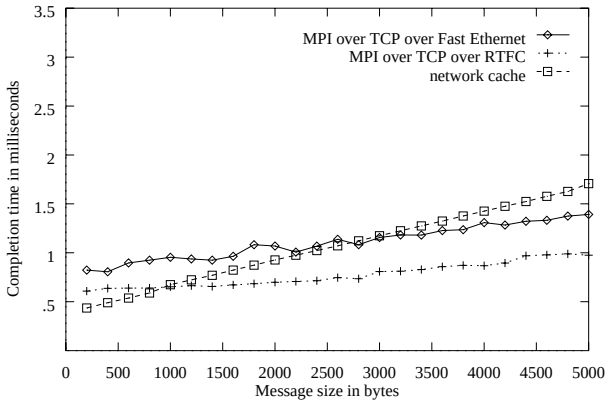

Fig. 5. Completion time of collective calls for broadcast, two nodes

Figure 5 gives the measurements for the broadcast operation between two nodes. With two processes the messaging reduces to sends and receives. Figure 5 shows that MPI performs well over TCP over RTFC in comparison to MPI over Fast Ethernet. The network cache implementation performs well for very small messages, but then poorly as the message size increases. This is expected, since with only two nodes involved in the messaging there is less advantage in using a broadcast network. There is some advantage in using network cache messaging in the case of small messages, but the prototype code for collective operations via network cache uses a separate step for distributing addresses and signalling that the buffer is free, even in the case of two processes. This extra step causes the network cache messaging to perform worse than the MPI over TCP over RTFC equivalent in the case of two nodes as the message size increases.

Figure 6 illustrates the measurements for the broadcast operation between three nodes. Again, MPI performs well over TCP on RTFC in comparison to MPI on Fast Ethernet. This graph shows noticeable changes in slope in the MPI results at packet boundaries of approximately 1500, 3000, 4500, etc. The same

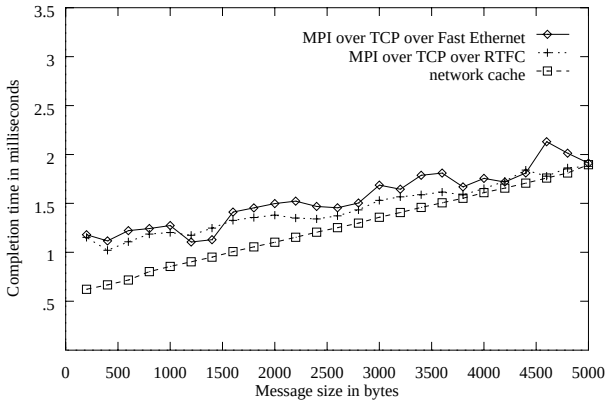

Fig. 6. Completion time of collective calls for broadcast, three nodes

buffer size is used internally for TCP/IP for both Fast Ethernet and RTFC. In the case of three processes, one per node, the network cache implementation does have a relative advantage over MPI messaging over TCP. Messaging via writes to network cache is not sensitive to packet boundaries and the completion time is smaller than the completion time for MPI over TCP.

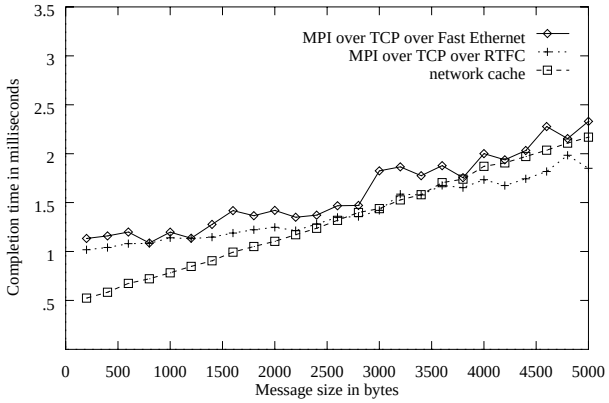

Fig. 7. Completion time of collective calls for gather, three nodes

Figure 7 illustrates the measurements for the gather operation between three nodes. As before, Figure 7 shows that MPI performs well over TCP over RTFC in comparison to MPI over Fast Ethernet. Also, as in the case of broadcast for three processes, one per node, the network cache implementation does have a relative advantage over MPI messaging over TCP. Figure 8 illustrates the measurements for the scatter operation between three nodes, with similar results. Graphs for

the other collective operations also show similar results but are not included here due to space limitations.

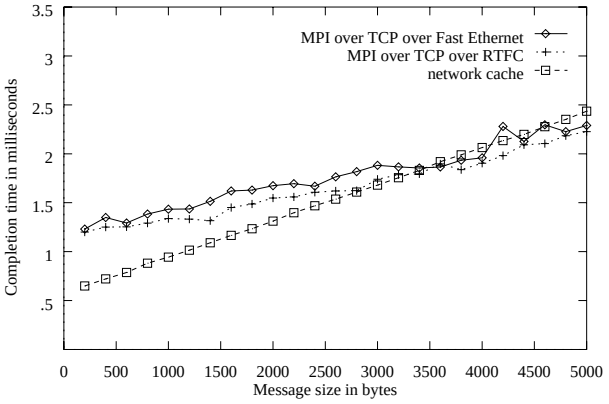

Fig. 8. Completion time of collective calls for scatter, three nodes

The relatively higher rise in the line for messaging via writes to network cache is noticeable in each of the graphs and is the subject of further study. This effect may be due to the combination of the beta nature of the current hardware and software platform and the experimental synchronization technique. In the current platform, a data transfer between host memory and network cache on the interface card must complete before control returns to the user application, even in the case of DMA access. As a result, in the current setup, notification of a sent message begins after the completion of the network cache write. In contrast, the TCP drivers on both platforms move the message in units of a framesize at a time, and allow for return to the user application while transfer is still being completed. This overlap gives greater efficiency for TCP over RTFC for larger message sizes as compared to direct network cache writes in the current setup. Effective use of DMA access to the network cache memory will be provided in the full implementation of the AmpDC middleware and final RTFC product and will help to eliminate this source of inefficiency.

6 Conclusions and Future Work

The intent of this work is to investigate the performance of Real-Time Fiber Communications and its applicability as an interconnection technology for parallel computing. MPI over TCP on RTFC performs well, and better than MPI over TCP on Fast Ethernet. As in the case of other gigabit-speed networks, the improvement in performance is limited by the use of high-overhead protocols such as TCP.

Improvements for parallel collective operations may be obtained using the network cache architecture. The broadcast, multicast, and simple hardware distributed shared memory mechanisms provide the potential for improvement of traditional MPI implementations of collective operations. The improvement in performance is currently limited by the experimental design for collective communication in that it uses a separate messaging step for synchronization. In contrast, a full handshake synchronization technique is completely eliminated with a full RTFC implementation. In particular, effective use of DMA access and the availability of interrupt messages to remote nodes will allow these collective operations to be easily and efficiently implemented over RTFC.

One of the goals of this project is to create an implementation of MPI for network cache. Future efforts also include the further evaluation of methods for using the network cache as a platform for message passing. Related work includes study of the implementation and measurement of efficient collective communications and multicast mechanisms [3]. The RTFC Team is exploring many of the features and capabilities of RTFC, including the use of the AmpDC environment as a parallel programming environment, bounds and limitations of the RTFC rostering algorithm, and new techniques for flow control.

Acknowledgement is given to Larry Wilbur, Dave Belo, Mike Jeffers, Glynn Smith, and Joan Clark of Belobox Systems, Inc. and to Joe Gwinn of Raytheon Company for their helpful suggestions and indispensable support on this project.

References

1. Bahaa-El-Din, W. H., and Liu, M. T. Register insertion: A protocol for the next generation of ring local-area networks. *Computer Networks and ISDN Systems 24* (1992), 349–366.
2. Belobox Systems Inc., AmpDC Advanced Multiprocessor Distributed Computing Environment, 1997. http://www.belobox.com/ampdc.html.
3. Dunigan, T. H., and A.Hall, K. PVM and IP multicast. Tech. Rep. ORNL Report ORNL/TM-13030, Oak Ridge National Laboratory, December 1996.
4. Gropp, W., Lusk, E., Doss, N., and Skjellum, A. A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard. Tech. Rep. Preprint MCS-P567-0296, Argonne National Laboratory, March 1996.
5. Gwinn, J. RTFC Principles of Operation, RTFC-2001. Tech. rep., Belobox Systems Inc., Irvine, California, January 2 1998.
6. Martin, R. P. HPAM: An active message layer for a network of HP workstations. In *Hot Interconnects II* (August 1994).
7. Nupairoj, N., and Ni, L. Benchmarking of multicast communication services. Tech. Rep. Technical Report MSU-CPS-ACS-103, Department of Computer Science, Michigan State University, April 1995.
8. Rajagopal, P. Study of the network cache architecture as a platform for parallel applications. Master's thesis, Vanderbilt University, August 1998.
9. RTFC-team. RTFC Micropacket Data Link Layer, RTFC-2002. Tech. rep., Belobox Systems Inc., Irvine, California, October 1997.

This article was processed using the LaTeX macro package with LLNCS style