Performance Evaluation of Group Communication Architectures in Large Scale Systems using MPI

Kayhan Erciyes[1], Orhan Dagdeviren[1], and Reşat Ümit Payli[2]

[1] Izmir Institute of Technology
Computer Eng. Dept., Urla, Izmir 35430, Turkey
{orhandagdeviren, kayhanerciyes}@iyte.edu.tr
[2] Computational Fluid Dynamics Laboratory
Purdue School of Engineering and Technology
Indiana University-Purdue University
Indianapolis, Indiana 46202, U.S.A.
rpayli@iupui.edu

Abstract. Group communication is an important paradigm for fault tolerance in large scale systems. We describe various group architectures as pipelined, hierarchical, daisy and hypercube groups each consisting of separate clusters, investigate the theoretical performance bounds of these architectures and evaluate their experimental performances using MPI group communication primitives. We first derive time bounds for multicast message deliveries in these architectures and then provide tests to measure the times taken for the same operation. The multicast message delivery times are tested against the number of clusters within a group and the size of the multicast message. We conclude that daisy architecture is favorable both in terms of delivery times and message sizes theoretically and experimentally.

1 Introduction

A group is a logical name for a set of computing elements whose membership may change with time. Replication using process groups for fault tolerance has attracted many researchers for many years [1–3]. There are several systems which provide fault tolerant group communication such as Transis [4], Horus [5] and Totem [6]. Moshe [7] extends these services to a WAN. The common goal of these projects is to provide a reliable multicast communication for process groups.

MPI is a library specification for message passing, proposed as a standard by a broadly based committee of vendors, implementors and users [8, 9]. *MPI* provides enhanced group communication primitives [10]. An *MPI* communication operation always specifies a communicator. This identifies the process group that is engaged in the communication operation and the context in which the communication occurs. In this study, we investigate and evaluate Group Communication primitives in the pipelined, daisy, hierarchical and hypercube architectures using

MPI. Background on process groups and *MPI* Group Communication primitives is reviewed in section 2. Sections 3, 4, 5, 6 briefly describe the architectures with the test results obtained. Finally a comparison of the results with the discussions is presented in Section 7.

2 Background

2.1 Group Communication

Replication is a common approach to achieve fault tolerance in a distributed system such that replicas provide redundancy in case of a failure of a server. Two main classes of replication are the *active* and *passive* replications. In passive replication, client deals only with one replica and the primary sends messages to the secondaries to update their views. A client sends a message to all of the replicas in active replication and the states of the replicas are maintained as identical, in general, using finite state machines. To ensure consistency of the replicas, a group communication primitive called the Total Order Multicast may be used which guarantees that the requests by the clients are received by all replicas in the same order. The group management module should also provide the two primitives; *send_multicast* to send a message to all members and *receive_multicast* to receive a message sent by a member of the group. These two primitives can be realised using various approaches such as *reliable broadcast, reliable FIFO broadcast and total order multicast. Reliable Broadcast* of a message in a group ensures that messages are delivered by all processes or none.

2.2 MPI Group Communication Primitives

In MPI, process groups allow a subset of processes to communicate among themselves using local process identifiers and to perform collective communication operations without involving other processes. The context forms part of the envelope associated with a message. A receive operation can receive a message only if the message was sent in the same context. Hence, if two routines use different contexts for their internal communication, there can be no danger of their communications being confused. All communication operations have used the default communicator *MPI _COMM_WORLD* , which incorporates all processes involved in an *MPI* computation and defines a default context. A set of group communication routines which are used for creating new groups and communicators, multicasting messages to all process in a communicator is defined in *MPI* is summarized below:

- *MPI_Comm_group* : Function *MPI_Comm_group (MPI _Comm comm, MPI _Group *group)* returns the process group associated with the communicator.
- *MPI_Comm_create* : The collective function *MPI_Comm_create (MPI_Comm old_comm, MPI_Group group, MPI_Group *new_comm*) creates a new communicator from the processes listed in *group*.

– MPI_Comm_split : Collective function *MPI_Comm_split (MPI_Comm old _comm, int partition, int new_rank, MPI_Comm *new_comm)* partitions the processes in an existing communicator(old_comm) into one or more subgroups.
– MPI_Group_incl : Function *MPI_Group_incl (MPI_Group old, int new_size, int *old_ranks, MPI_Group *new*) produces a new group from an existing group. The size of the the new group is specified by parameter new_size.
– $MPI\_Barrier$: $MPI\_Barrier(MPI\_Comm)$ is a collective communication function that performs a barrier synchronization among all processes in the specified communicator.
– MPI_Bcast : Function *MPI_Bcast (void *buffer, int cnt, MPI_Datatype dtype, int root, MPI_Comm comm)* is a collective communication operation allowing one process to broadcast a message to all other processes in a communicator [11].

3 Pipelined Architecture

Pipelined architecture is constructed by the serial arrangement of groups which includes equal size of processors. Each group has a leader which is responsible to send and receive messages from another group and to multicast messages to its group elements. A pipelined architecture with 24 processors and 4 groups is shown in Fig. 1. Each group has 6 processors in which 0, 6, 12 and 18 are the group leaders.

Pipelined architecture can be constructed by row wise partitioning of the *MPI_COMM_WORLD* communicator. Leaders of each group can send messages using *MPI_Send* and *MPI_Recv* primitives in *MPI_COMM _WORLD* communicator.

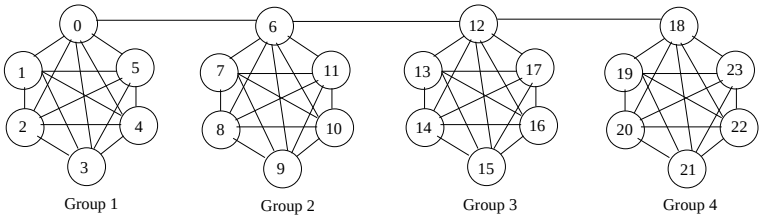

Fig. 1. Pipelined Architecture with 24 processors and 4 groups

Theorem 1. *Multicast communication in pipelined architecture takes* $\Theta(mk)$ *time where* m *is an upperbound on the number of clusters and* k *is an upperbound on the number of processors in a cluster,*

Proof. Assume for simplicity, each multicast operation within a cluster of the pipelined group will consist of k one-to-one communications. Each of these muticast communications is performed sequentially, one after each other along the

pipeline. Therefore, we would have a total of $m * k$ message deliveries within the clusters and $m-1$ messages among the clusters, resulting in a total of $mk+m-1$ steps to deliver the multicast message to all group members. The total time can therefore be approximated by $\Theta(m(k+1)-1) \sim \Theta(mk)$

We performed the tests on the 120 processors of the AVIDD cluster of Indiana University, U.S.A. AVIDD is an IBM eSeries cluster which consists of 96 nodes. Each node has 2 2.4GHz Intel Pentium 4 Xeon CPUs and a 2.5GB memory with the Myricom Myrinet interconnect. Peak performance of the cluster is 0.55 Tera Flops. One-to-all broadcast performance of pipelined group in this environement is measured with respect to message size and total group number. In Fig. 2, message size is varied from 4 bytes to 40K bytes, processor number is fixed at 24 and the total group count is varied between 2 and 8. A pipelined architecture with 8 groups results in the minimum mean time of 806 microseconds for a group multicast operation.

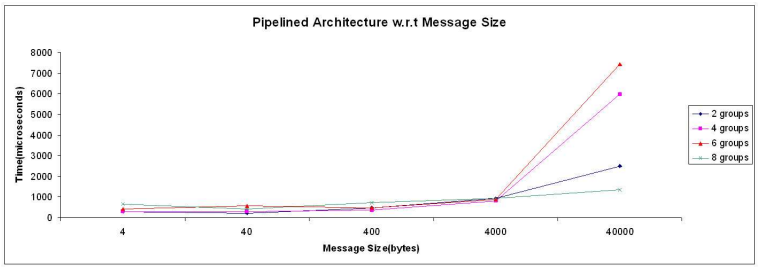

Fig. 2. Pipelined Architecture Multicast Run-times against Message Size

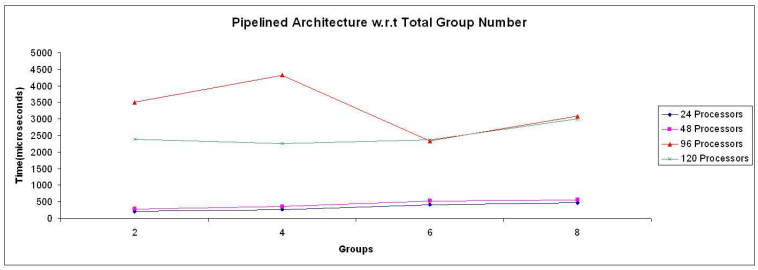

Fig. 3. Pipelined Architecture Multicast Run-times against Total Group Number

In Fig. 3 total group number is varied from 2 to 8 groups, message size is fixed at 4 bytes and the total processor number is varied from 24 to 120. Pipelined architectures with 24 and 48 processors seem to finish one-to-all broadcast op-

eration in approximate time values but a sharp increase occurs when we double 48 processors to 96 processors.

4 Daisy Architecture

Daisy architecture is constructed by a number of groups in which leaders are connected to form a group which overlaps with all other groups. Each group leader is responsible to multicast messages to its group. A daisy architecture with 24 processors and 5 groups are shown in Fig. 4.

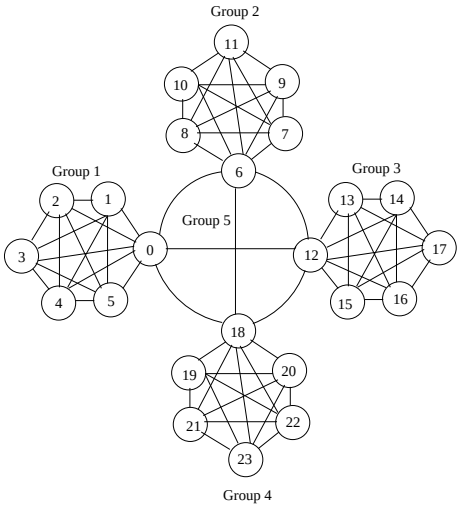

Fig. 4. Daisy Architecture with 24 processors and 5 groups

Theorem 2. *Multicast communication in the daisy architecture takes* $O(3k)$ *time where* k *is an upperbound on the number of processors.*

Proof. A multicast operation within a cluster of the daisy group will take k one-to-one communications to reach the central cluster. There will be k one-to-one mesages by the central cluster and another k mesages by the outer clusters which are performed in parallel resulting in a total of $O(3k)$ message times.

It should be noted that the message delivery time of the daisy architecture is independent of the number of clusters and the final delivery in the outer clusters are performed as parallel multicast communications. Daisy architecture can be constructed by row wise and column wise partitioning of the *MPI_COMM_WORLD* communicator. Column wise communicators except group leaders's communicator are destroyed by *MPI_Comm_free*. Experimental setup is the same as in pipelined test. A daisy architecture with 8 groups results

in the minimum mean time of 338 microseconds which is shown in Fig. 5. In Fig. 6 when we double the 48 processors a sharp increase occurs.

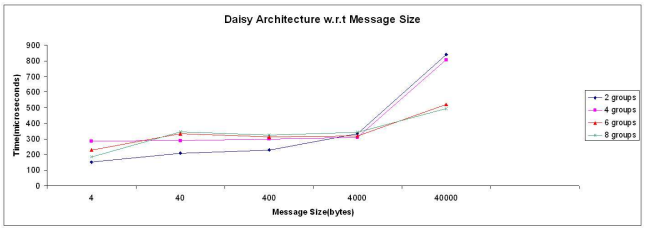

Fig. 5. Daisy Architecture Multicast Run-times against Message Size

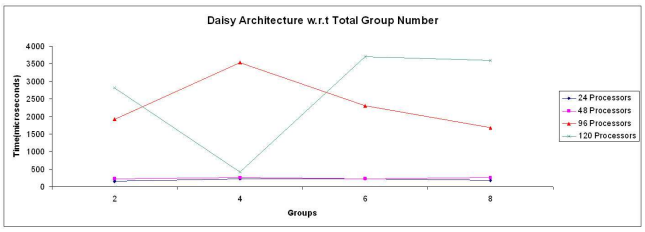

Fig. 6. Daisy Architecture Multicast Run-times against Total Group Number

5 Hierarchical Architecture

Hierarchical Architecture is constructed by different levels of groups which can contain different number of processors as in [12]. A hierarchical architecture with 24 processors and 7 groups is shown in Fig. 7. Level 1, Level 2 and Level 3 groups contains 3,4 and 4 processors respectively. Group leaders are responsible to exchange messages between upper level of connected groups. Processor groups can be created by MPI_Comm_group and MPI_Group_incl. Communicators of these groups can be created by MPI_Comm_create.

Theorem 3. *Multicast communication in the hierarchical architecture takes* $O(2kl)$ *time where* l *is an upperbound on the number of levels of hierarchy.*

Proof. In the worst case, it would take $2l - 1$ steps for a multicast message to reach the farthest node where l is the count of levels of the hierarchy. Since there are k nodes in each cluster, there would be $O(2k(l-1)) \sim O(2kl)$ unicast messages to accomplish a multicast communication in the worst case.

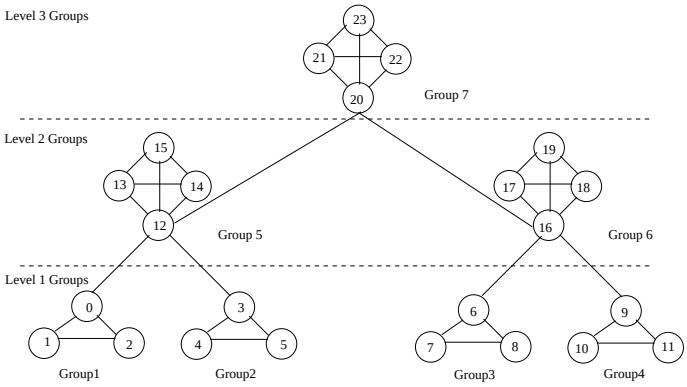

Fig. 7. Hierarchical Architecture with 24 processors and 7 groups

Corollary 1. *For the special case of a binary tree of the hierarchical architecture, multicast communication takes* $O(2k \log m)$ *time.*

Proof. For the binary tree of the hierarchical architecture, there would be a total of $2^l - 1$ tree nodes meaning $m = 2^l - 1$. We can therefore approximate $l = \log m$. The total count of multicast communication steps is $2l - 1$. Each multicast communication has k unicast communications resulting in a $O(k * (2l - 1))$ steps for the total multicast. Substituting for l in the complexity equation yields $\sim O(2k \log m)$ for the total time of multicast in the binary tree in the worst case.

In Fig. 8 one-to-all broadcast times are measured against message size on a binary tree with 24 processors and 7 groups and 1625 microseconds is the mean time. In Fig. 9 one-to-all broadcast times are measured with respect to total processor number with fixed 7 groups and 4 bytes messages. Processor number is varied between 24 and 96. A significant increase happens when we double the processor number to 96 processors.

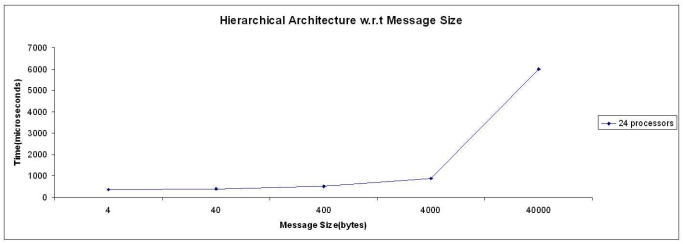

Fig. 8. Hierarchical Architecture Multicast Run-times against Message Size

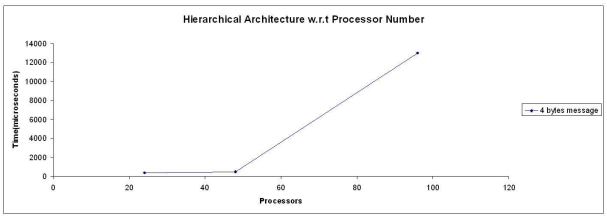

Fig. 9. Hierarchical Architecture Multicast Run-times against Total Processor Number

6 Hypercube Architecture

A hypercube architecture is a widely used architecture for parallel/distributed applications [13]. A two-dimensional hypercube of four processors is constructed from one-dimensional hypercubes by connecting corresponding nodes. In general a $d-$dimensional hypercube is constructed by connecting corresponding nodes of two $(d-1)$ dimensional hypercubes. A hypercube architecture with 24 processors and 8 groups is shown in Fig. 10. One-to-all broadcast in hypercube topology can be implemented by modifiying the procedure in [14].

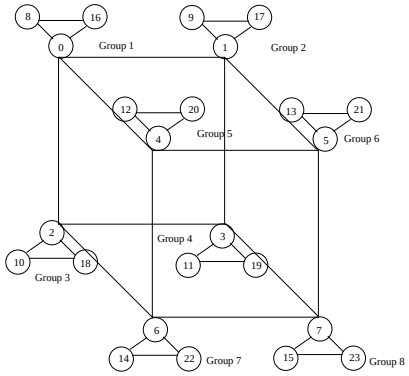

Fig. 10. Hypercube Architecture with 24 processors and 8 groups

Theorem 4. *Multicast communication in the hypercube architecture takes* $O(k \log m)$ *time where* m *is an upperbound on the number of clusters (number of hypercube vertices) and* k *is an upperbound on the number of processors in the hypercube vertex cluster.*

Proof. A one-to-all communication in a hypercube of m nodes takes $\log m$ time. Since a one-to-all communication of $O(k)$ is performed within each hypercube vertex cluster prior to sending the message along another dimension, we have $O(k \log m)$ message times ignoring the times taken for the single message deliveries along the edges of the hypercube.

One-to-all broadcast performance in a hypercube architecture with 24 processors and 3 dimensions is obtained with respect to message size in Fig. 11 and a mean time of 1168 microseconds is measured.

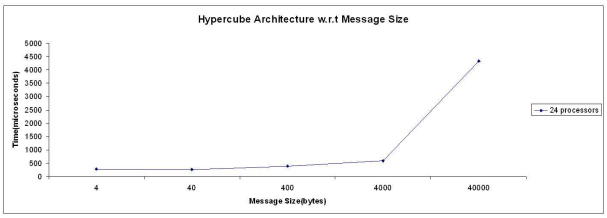

Fig. 11. Hypercube Architecture Multicast Run-times against Message Size

We also measured the one-to-all broadcast performance with respect to processor number and dimension in Fig. 12. A hypercube architecture with 3 dimensions can be realized with 24, 48, 96 and 120 processors, 4 dimensions can be realized with 48 and 96 processors and 5 dimensions can be realized with 96 processors. A sharp increase can be seen when we double the 48 processors in Fig. 12.

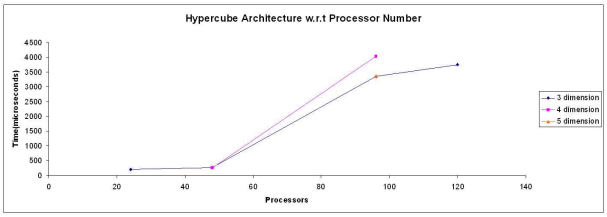

Fig. 12. Hypercube Architecture Multicast Run-times against Total Processor Number

7 Discussions and Evaluations

We evaluated various group communication architectures in a cluster of 120 processors using MPI. The groups in the tests were connected in pipelined, daisy, hierarchical and hypercube architectures. In Fig. 13, architectures are compared with respect to message size with a constant total number of processors of 24. In Fig. 14, architectures are compared with varying total number of processors.

The comparison of message complexities and the message delivery times of these architectures are displayed in Table 1. It can be seen that the daisy architecture provides the minimum multicast message delivery times for both the

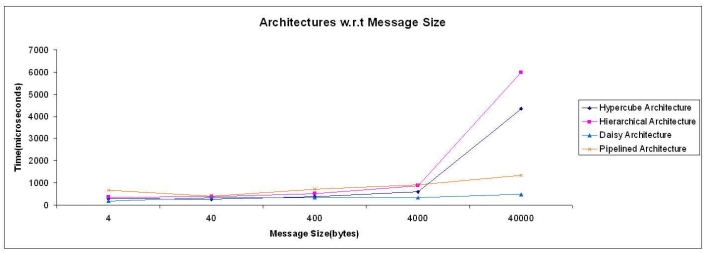

Fig. 13. Architectures Multicast Run-times Comparison against Message Size

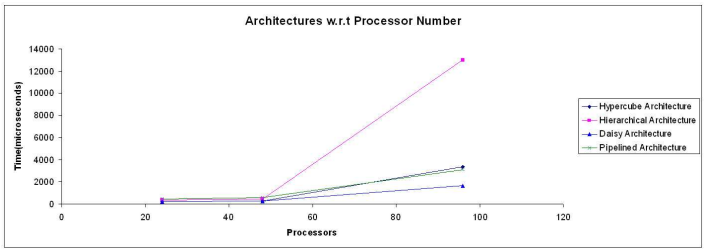

Fig. 14. Architectures Multicast Run-times Comparison against Total Processor Count

fixed number of processors and the fixed size of message cases as expected. Theoretically, the next best performance should have been the hypercube and the pipelined architecture should have provided the worst performance but the experimental results did not wholly support this view. Pipelined architecture resulted in similar performance to the hypercube. A larger cluster count may affect the results. Based on the analysis, we expected that the binary tree architecture should have twice longer times than the hypercube which conforms to the tests.

Table 1. Comparison of the Group Architectures

Topology	Time Complexity	Fixed Proc.(t) (μs)	Fixed Msg. Size(t) (μs)
Pipelined	$O(km)$	1776	806
Daisy	$O(3k)$	1429	338
Hierarchical(BT)	$O(2k \log m)$	4610	1625
Hypercube	$O(k \log m)$	1888	1168

Our general conclusion is that the daisy architecture is favorable for multicast communication using *MPI* in terms of message delivery times and message sizes. We are planning to use the daisy architecture based group communication in the Grid architecture [15] where nodes do fail frequently and hence fault tolerance using replication is an important paradigm to provide a reliable service.

References

1. Birman K. P., van Renesse, R., Reliable Distributed Computing with the Isis Toolkit, IEEE Computer Society Press, Los Alamitos, Ca., (1994).
2. Chockler, G, Keidar, I., Vitenberg, R., Group communication specifications: a comprehensive study, ACM Computing Surveys, (2001), 33(4) , 427-469
3. Cristian F., Synchronous and Asynchronous Communication, Communications of the ACM. Special Section on Group Communication, 1996, 39(4)
4. Y.Amir et all, Transis: A communication subsystem for high availability. Proc. of 22nd IEEE Int'l Symp. on Fault-Tolerant Computing, IEEE Press, NJ, 76-84
5. Van Renesse R., Birman K. P.,Maffeis S., Horus : A Flexible Group communication System, CACM, Special sect. on Group Comm., (1996), 39(4)
6. Y.Amir et all, The TOTEM Single Ring Ordering and membership Protocol, ACM Trans. Comp. Systems., 1995, 13(4)
7. Keidar, I. et al, Moshe: A group membership service for WANs, ACM Transactions on Computer Systems (TOCS) (2002), 20(3) , 191-238
8. Gropp, W., Lusk, E., Doss N. and Skjellum, A. : A High-Performance, Portable Implementation of the MPI Message Passing Interface Standard, MPI Developers Conference, (1995)
9. Squyres, J. M., Lumsdaine, A., George, W.L., Hagedorn, J.G. and Devaney, J.E., : The Interoperable Message Passing Interface (IMPI) Extensions to LAM/MPI, MPI Developers Conference, Ithica, NY, (2000)
10. Yuan, X., Daniels, S., Faraj, A., Karwande, A. : Group Management Schemes for Implementing MPI Collective Communication over IP Multicast, The 6th Int. Conf. on Computer Science and Informatics, Durham, NC, (2002), 8-14
11. Quinn, M. J., : Parallel Programming in C with MPI and OpenMP, International Edition, Mc Graw Hill, (2003).
12. Tunali, T, Erciyes,K., Soysert, Z.: A Hierarchical Fault-Tolerant Ring Protocol For A Distributed Real-Time System, Special issue of Parallel and Distributed Computing Practices on Parallel and Distributed Real-Time Systems, (2000), 2(1), 33-44
13. Allahverdi, N, Kahramanli, S, Erciyes,K. : A Fault Tolerant Routing Algorithm Based on Cube Algebra for Hypercube Systems , JSA, 2000, 46(2), 201-205
14. Grama, A., Gupta, A., Karypis, G., Kumar, V. : Introduction to Parallel Computing, Second Edition, Addison Wesley Longman, Inc., (2003).
15. Foster, I., Kesselman, C., Tuecke, S.: The Anatomy of the Grid: Enabling Scalable Virtual Organizations. Int. Journal of High Performance Computing Applications, (2001), 15(3), 200-222