Topology-Based Hypercube Structures for Global Communication in Heterogeneous Networks[1]

Silvia M. Figueira[1] and Vijay Janapa Reddi[2]

[1] Department of Computer Engineering, Santa Clara University
Santa Clara, CA 95053-0566, USA, sfigueira@scu.edu

[2] Department of Computer Engineering, University of Colorado, Boulder
Boulder, CO 80309, USA, janapare@colorado.edu

Abstract

Hypercube structures are heavily used by parallel algorithms that require all-to-all communication. When communicating over a heterogeneous and irregular network, the performance obtained by the hypercube structure will depend on the matching of the hypercube structure to the topology of the underlying network. In this paper, we present strategies to build topology-based hypercubes structures. These strategies do not assume any kind of topology. They take into account the communication cost between pair of nodes to provide a performance-efficient hypercube structure. These enhanced hypercube structures help improve the performance of parallel applications that require all-to-all communication in heterogeneous networks by up to ~30%.

1 Introduction

Hypercube structures allow a computation that requires all-to-all communication among P tasks to be performed in log$P$ steps. All-to-all communication is used by a variety of parallel algorithms, such as barrier synchronization, vector reduction, matrix multiplication, and sorting, making the hypercube one of the most useful structures in parallel computation [7]. In fact, many popular parallel algorithms use a hypercube communication structure, as shown by Leighton [17]. Some examples are the fast Fourier transform [19, 25], parallel prefix [24], and various computer vision [24] and linear algebra computations [13]. In addition, the MPICH's implementation [22] of the barrier synchronization operation is based on hypercubes.

To use a hypercube structure in an all-to-all operation, the processes are organized in a hypercube, i.e., the processes are assigned to positions in the hypercube structure and communicate with the nodes assigned to neighbor positions only. In homogeneous clusters, this assignment is generally done according to their nodes' identifiers, which are usually assigned *blindly* (independently of any performance measure) to the nodes [7, 11, 22]. For example, consider a cluster formed by 4 nodes located in the same LAN. These nodes are assigned identifiers 0, 1, 2, and 3. According to these identifiers, a 2D hypercube can be created, as shown in Figure 1, where node 0 communicates with nodes 1 and 2, but not to node 3, which also communicates with nodes 1 and 2.

[1.] This research was supported in part by NSF cooperative agreement ACI-9619020 through computing resources provided by the National Partnership for Advanced Computational Infrastructure at the San Diego Supercomputer Center.

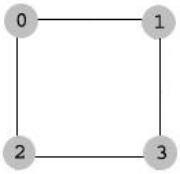

Fig. 1. 2D hypercube formed according to the nodes' identifiers.

When using a hypercube structure, communication takes place between nodes that are connected in the hypercube. For this reason, the organization of the hypercube is a key process in the all-to-all communication. In regular-topology platforms, organizing the hypercube blindly may lead to good performance. However, in heterogeneous, irregular networks, we need a more sophisticated strategy to build the hypercube. Intuitively, the algorithm would execute more efficiently if we could form the hypercube by having communication take place only between nodes that are separated by a low-communication cost path, as shown in [6] for binomial trees.

This paper presents possible solutions to embed a given network topology into a hypercube template to be used in all-to-all communication. We evaluate the efficiency of the strategies proposed by comparing both the cost of the hypercubes generated and the time to execute a barrier-synchronization operation using the different hypercubes. The strategies discussed do not assume any kind of topology, i.e., the machines may be organized in any topology. It takes into account the communication cost between the nodes to come up with a performance-efficient hypercube to be used by the barrier-synchronization algorithm.

The strategies proposed are independent of the measure used as communication cost. In our experiments, latency is used but, in fact, the measure to be used should reflect the application characteristics. For example, for an application that sends a large number of short messages, the communication cost should reflect the latency. For an application that sends a small number of large messages, the communication should reflect the bandwidth. For an application that sends a large number of large messages, the communication cost should reflect both the latency and the bandwidth which, combined with the message size used by the application, provide an accurate model for communication cost. Note that these costs can be obtained with a performance prediction tool, such as the Network Weather Service [27].

This paper is organized as follows. Section 2 discusses related work. Section 3 presents the hypercube-based all-to-all communication algorithm used in the paper. Section 4 presents algorithms for creating topology-based hypercubes to be used in all-to-all communication operations. Section 5 shows experiments performed and results obtained. Section 6 concludes.

2 Related Work

Hypercube structures play an important role in global communication operations, and have been the subject of several research papers. Bertsekas et al have described optimal communication algorithms for hypercubes [4]. In [3], the authors presents a study on algorithms for collective operations for homogeneous parallel environments, and embeddings of different structures into hypercubes have been the subject of [5, 12, 18, 26].

In [1], the authors deal with heterogeneity by forming broadcast trees according to the

capacity of each machine. In [2], the authors present a communication model of heterogeneous clusters for performance characterization of collective operations.

Several groups have been working on projects related to global communication in hierarchical topologies. In [20], the authors present ECO, a packet containing efficient collective operations for interconnected clusters. ECO groups hosts according to the network topology, i.e., each group contains nodes that belong to the same (homogeneous) cluster. Based on these groups, ECO implements the collective operation using a specific algorithm for each LAN. In [16], the authors also present a solution for more efficient global communication in hierarchical networks of workstations. They also group the hosts according to the network topology, but they use a binomial tree for each LAN. In [15], the authors describe MagPIe, a communication library that uses performance-efficient structures for global-communication operations in GRID environments. Global communication in WAN environments, such as the GRID, have been the subject of many other papers [8, 14], which employ a hierarchical structure to reflect the hierarchy existent in these environments.

PVM (Private Virtual Machine) [11] and MPICH [22], which is an implementation of MPI (Message Passing Interface) [21], are libraries used in practice by scientific applications using heterogeneous networks. Both of them provide various collective communication operations, such as one-to-all and all-to-all. Originally, they did not take the network topology into account. However, the Globus group have proposed the enhancement of MPICH to accommodate for the hierarchical structure of GRID environments. They use a hierarchical structure, as discussed in [9].

3 Hypercube-Based Communication Algorithm

In [7], Foster proposes a hypercube-based algorithm for all-to-all communication. It uses a hypercube communication template. The algorithm is executed by each task in a hypercube communication structure (obtained by the processes' identifiers). This algorithm allows an operation that requires all-to-all communication among *P* processes to be performed in *logP* steps. The algorithm is presented below:

```
procedure hypercube (myid, input, logp, output)
begin
    state = input
    for i = 0 to logp - 1
        dest = myid XOR 2^i
        send state to dest
        receive message from dest
        state = OP (state, message)
    endfor
    output = state
end
```

The value *logp* represents the size of the hypercube, and *myid* represents the node's identifier. XOR denotes an exclusive OR operation, and OP is the user-supplied operator, used to combine local data with data arriving from the *i*th neighbor in the hypercube. In each step of the algorithm, each process exchanges its local *state* (which embeds its local *input* with the information received so far from its neighbors) with one of its neighbors in the hypercube and, then, combines the *message* received from that neighbor with *state* to generate a new *state*. Note that, at each step, each node communicates with the neighbor indicated by *dest* = *myid* XOR 2^i, which does not depend on the network topology.

As shown in [7], this algorithm can be used efficiently, in regular-topology platforms, for vector reductions, matrix transpositions, merge sorts, and so on. However, as shown in

Section 5, in heterogeneous networks, the algorithm's performance depends on the organization of the hypercube and, at each step, *dest* should be a node selected according to the topology of the network.

4 Enhancing Hypercube Structures

Our strategies to reorganize the nodes to form a more performance-efficient hypercube are based on the communication cost between the nodes. The hypercube-based algorithm works synchronously and, at each step, each node communicates with a specific node in the same subcube. In this case, placing nodes that are connected by a low-cost path in communicating positions, so that communication in each step of the algorithm uses a low communication-cost path, will decrease communication costs and lower the total cost of the all-to-all operation.

The following subsections present our three algorithms developed to provide topology-based hypercubes. The algorithms take the communication cost between pairs of nodes into account to form a performance-efficient hypercube.

4.1. Dim2_Cube

This algorithm tries to optimize the first dimension of the hypercube to decrease the cost in the first step of the all-to-all operation. The algorithm, which is shown below, is based on the following procedure: For every even position *i*, assign the first unmarked node *n* and select *n*'s closest unmarked node to be placed at *i*'s neighbor position in the first dimension.

```
Dim2_Cubes ( )
begin
    unmark all nodes
    for i = 0, 2, 4, ..., N-2
        n = 0
        while n is marked
            n = n + 1
        end while
        assign node n to position i
        mark node n
        j = non-used closest node to n
        make node j the first neighbor of node i
        mark node j
    end for
end
```

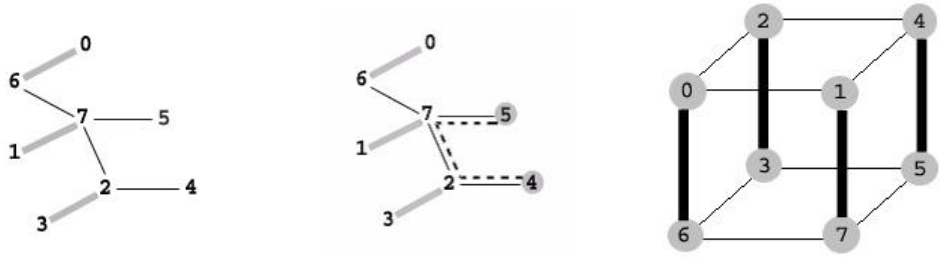

Fig. 2. Following the Dim2_Cube algorithm.

Figure 2 shows an example of a hypercube obtained with the algorithm above. Figure 2 (left) shows a network in which every link has the same cost. Figure 2 (left) illustrates the first three steps of the algorithm: node 0 is placed in position 0 with node 6 as first neighbor, node 1 is placed in position 2 with node 7 as first neighbor, and node 2 is

placed in position 4 with node 3 as first neighbor. Figure 2 (middle) shows the last step of the algorithm, in which node 4 is placed in position 6 with node 5 (the last unmarked node) as first neighbor. Figure 2 (right) shows the hypercube obtained. Note that three out of the four edges in the first dimension of the hypercube shown in Figure 2 (middle) are optimum. However, the last edge obtained compromises the efficiency of the hypercube.

This algorithm executes in time $O(n^2)$, where n is the number of nodes and, because it focuses on the first dimension only, the gains obtained by the hypercubes generated are limited (see Section 5).

4.2. TSTS_Cube

This algorithm uses a Traveling Salesman Tour with Shortcuts (TSTS), produced from a Minimal Spanning Tree (MST), to generate a *line* of nodes, from which each node is assigned to the corresponding position of the hypercube, according to the gray code [23].

The Minimal Spanning Tree (MST) is a tree that contains all the nodes in a network, and it is formed in such a way that the cost from the root to each of the nodes is minimum. The TSTS can be obtained from the MST by traversing it, visiting every node just once. A detailed explanation of how to create a TSTS from an MST is found in [10], which proves that the TSTS obtained has a length that is at most twice the length of the optimum tour.

The algorithm presented below traverses the hypercube positions using the gray code [23] and assigns nodes to each position according to the TSTS, generated from the MST corresponding to the network topology. The MST provides nodes that are close to each other, using locality to benefit the hypercube.

```
TSTS_Cube ( )
begin
    generate the MST and the TSTS
    h = 0, is the first position in the hypercube
    n = 0, is the first node in the TSTS
    for i = 0 to N-1
        assign node n to position h
        h = next position in the hypercube
        n = next node in the TSTS
    end for
end
```

The figures below illustrate the TSTS_Cube algorithm. The algorithm generates the MST from a given graph, as the one shown in Figure 3 (left). Starting with position 0, follow the gray code indexes and the nodes in the TSTS, assigning each node to the corresponding position. In Figure 3 (middle), the numbers on the edges indicate the sequence in which the nodes are visited, according to the TSTS order. Each node visited is placed in the next consecutive position, according to the gray code. Nodes are selected until the hypercube is complete, as shown in Figure 3 (right).

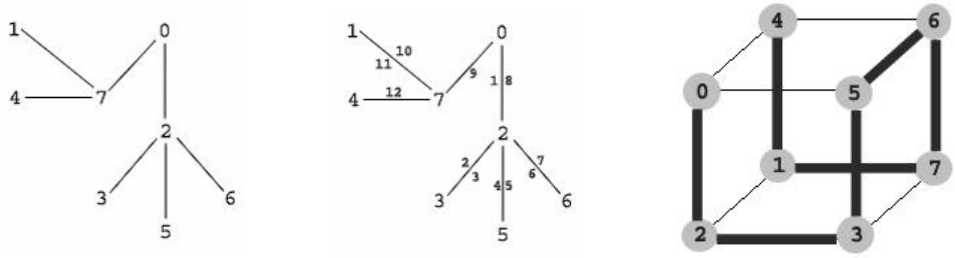

Fig. 3. Following the TSTS_Cube algorithm.

This algorithm executes in time $O(n^2)$, where n is the number of nodes and, even though the MST provides the algorithm with locality, this algorithm does not scale well. The results show that the hypercubes generated achieves an average gain of about 10% over the hypercubes generated blindly for up to 16 nodes, but the gain decreases as the number of nodes increase (see Section 5). This happens because nodes that are part of a branch of the MST (which means they are connected by a low communication-cost path), will be spread in a line and the locality provided by the MST will be lost. Also the TSTS does not reflect the MST perfectly because of the detours.

4.3. Eff_Cube

The previous algorithms are based on optimizing each neighbor only. However, due to the strong coupling in hypercubes, this may not be the best approach, particularly for large networks. For example, if a node C is a neighbor of both nodes A and B's, it may be a good neighbor for node A, but may not necessarily be a good neighbor for node B. This strong coupling between neighboring nodes and their edges leads us to think in terms of selecting neighbor nodes based on the overall efficiency of the selected node.

The Eff_Cube algorithm takes into account the *local cost* before assigning a node to a position. Thus, for a given location, the node that generates the least local cost (sum of the weights on the edge in each dimension) will be chosen for a given position.

The algorithm works as follows. Neighbors of position 0 are consecutively assigned nodes 0, 1, ..., *d*, where *d* is the maximum dimension of the hypercube, since neighbors for these positions have not been assigned yet. Then, starting with the next consecutive position, the algorithm traverses all its dimensions in search of empty neighbors. Let Y be the current position, and Y.n be its *n*th neighbor. The algorithm searches for a node X from the list of unused nodes that gives the least local cost. If the cost of the edges between pairs of nodes are $(X \leftrightarrow Y.0) = C_0$, $(X \leftrightarrow Y.1) = C_1$, $(X \leftrightarrow Y.2) = C_2$, ..., and $(X \leftrightarrow Y.n) = C_n$, then the local cost to be minimized is $C_0+C_1+C_2+...+C_n$. The algorithm assigns neighbors to the current node until all adjacent neighbors of the current index are filled with optimum nodes.

```
Eff_Cube ( )
begin
    for i = 0 to last position
        for j = 0 to last dimension
            neighbor position = position i's jth neighbor
            if neighbor position is empty
                minimum weight = infinite
                for k = 0 to last node
                    if node k has not been assigned yet
                        weight = 0
                        for d = 0 to last dimension
                            if a node was assigned the neighbor position's dth dimension
                                child = node in the dth dimension of neighbor position
                                weight = weight + cost between node k and child
                            end if
                        end for
                        if weight < minimum weight
                            minimum weight = weight
                            temp neighbor node = k
                        end if
                    end if
                end for
                position i's jth neighbor = temp neighbor node (assign temp neighbor node to neighbor position)
            end if
        end for
    end for
end
```

The figures below show an example of the Eff_Cube algorithm executing on a 3D hypercube. All neighbors of the starting position 0 are assigned nodes 0, 1 and 2. (Figure 4, left). The algorithm proceeds to position 1. For dimension 0, the algorithm starts traversing through the list of unused nodes to find the best suitable node. All nodes besides 0, 1 and 2 are checked as they are unused. Selection of the node is shown in Figure 4 (middle), Figure 4 (right), and Figure 5 (left), using as an example nodes 3, 5 and 7. For node 3 (Figure 4, middle), (3 ↔ 0) = 13, (3 ↔ 1) = 12, (3 ↔ 2) = 0, and the local cost = 25. For node 5 (Figure 4, right), (5 ↔ 0) = 11, (5 ↔ 1) = 9, (5 ↔ 2) = 3, and the local cost = 23. For node 7 (Figure 5, left), (7 ↔ 0) = 10, (7 ↔ 1) = 4, (7 ↔ 2) = 3, and the local cost = 17. Of all these nodes, node 7 yields the lowest local cost. Therefore, node 7 is chosen to be placed in the current empty index. All adjacent neighbors (Figure 5, right), dimension after dimension, are inserted in the same way, with the node that provides the least local cost.

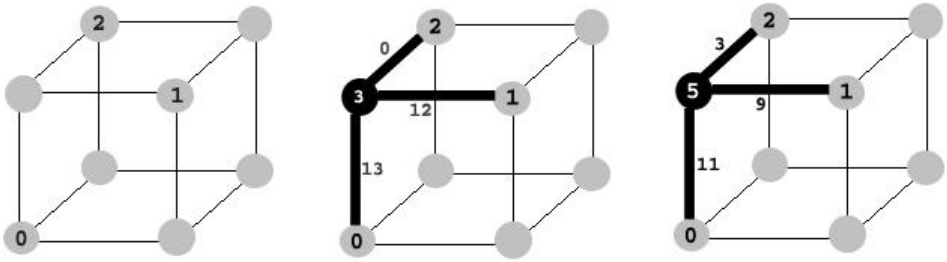

Fig. 4. Following the Eff_Cube algorithm.

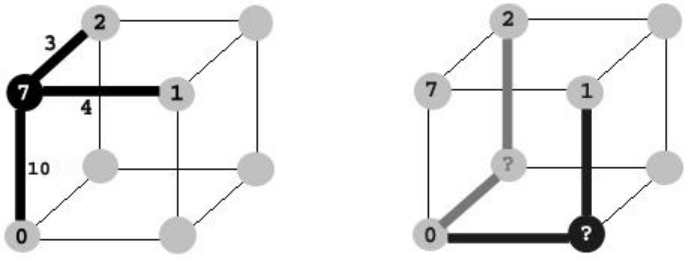

Fig. 5. Following the Eff_Cube algorithm.

This algorithm executes in time $O(n^2(logn)^2)$, where n is the number of nodes, and the gains obtained by the improved hypercubes is on average around 30% for large clusters (with 1024 nodes). This algorithm scales well. In fact, the average gain starts at about 10% for 8 nodes and increases consistently until 1024 nodes. Section 5 shows a representative set of the experiments performed.

5 Results

To show the effectiveness of the algorithms proposed, we have executed two kinds of experiments. First, we compare the cost of topology-based hypercubes with the cost of hypercubes obtained blindly by the system. This comparison is made for a series of topologies generated randomly. Then, we compare the time to execute a barrier synchronization operation using topology-based hypercubes with the time to execute the same operation

using blindly generated hypercubes. This comparison is also made for a series of topologies generated randomly.

5.1. Comparing Costs

The cost of a hypercube, which is the measure used in the comparison, is given by the cost of the node with the maximum cost, where the cost of each node is calculated as the sum of the weights between the node and each neighbor. Note that the cost for each node is calculated dimension-by-dimension, and that delays in previous dimensions, in which neighbors may have higher costs to their own neighbors, must be incorporated in the calculation of the cost for each dimension. The algorithm is shown below:

```
calc_cost ( )
begin
    for each node
        cost of node = weight between the node and the neighbor in the 1st dimension
    for i = 2nd to last dimension
        for each node n
            cost of n = max (cost of n, cost of n's neighbor in the ith dimension)
        for each node n
            cost of n = cost of n + weight between n and its neighbor in the ith dimension
        end for
    end for
    cost of hypercube = cost of the node with the maximum cost
end
```

Figure 6 shows a representative set of experiments, in which the average gain in cost is obtained by each algorithm for a set of 1,000 random topologies for each number of nodes. The gain represents how much lower (in %) the cost of a hypercube created by the respective algorithm is in comparison with the cost of a hypercube created blindly for the same topology. In Figure 6 (left), the nodes in each topology are apart by at most 5 Fast Ethernet links, representing networks in which the nodes are close. In Figure 6 (right), the nodes in each topology are apart by at most 20 Fast Ethernet links, representing networks in which the nodes are not close together.

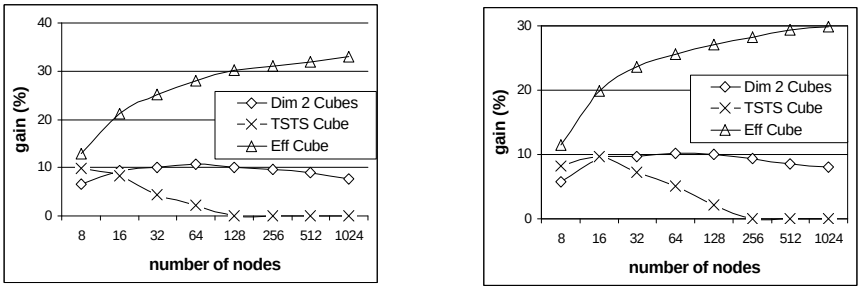

Fig. 6. Maximum cost between each pair of nodes is 5 links (left) and 20 links (right).

The graphs consistently show that the Dim2_Cube algorithm achieves an average of about 10% independently of the number of nodes, while the TSTS_Cube achieves a gain of about 10% for small networks, but does not scale well. The Eff_Cube algorithm achieves the highest gains and scales well, achieving an average gain of around 30% for 1024 nodes. These experiments show that the Eff_Cube algorithm implements the most promising approach, and that considering local costs leads to more efficient hypercubes. These graphs also show that the algorithms' behavior is independent of the maximum communication cost between the nodes.

5.2. Comparing Execution Times

Besides comparing costs of hypercubes, we have also executed experiments to compare the time to execute a barrier synchronization operation with the different hypercubes. The experiments were executed on the Blue Horizon IBM SP at the San Diego Supercomputer Center. The IBM-SP is a LAN-based, regular-topology cluster, in which we have emulated diverse heterogeneous networks by enforcing different latencies between the nodes. Our emulator was implemented using MPI [21]. Given the number of nodes and a maximum communication cost between any pair of nodes, the emulator generates random heterogeneous networks, which have various topologies. The emulator generates the topologies by generating the communication cost between every pair of nodes. Communication cost are in number of links. The latencies to be enforced are obtained by multiplying each random cost and a *base latency*, which is equivalent to the latency between two workstations connected by fast Ethernet (100Mbps), i.e., ~35ms. Each emulation executes 10,000 barrier synchronization operations.

Each graph below shows, for each algorithm, the gain obtained when comparing the improved hypercube with the one obtained blindly. For each algorithm, the topology used was the one that provided the best result among the experiments presented in Subsection 5.1. The graphs show the cost and the emulation time gains. Note that they match, showing that our cost calculation is accurate.

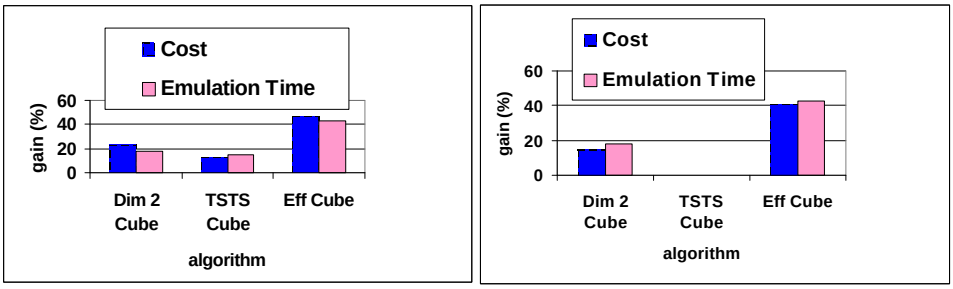

Fig. 7. 128 nodes (left) and 1024 nodes (right).

The Eff_Cube algorithm is the best option and achieves gains up to ~40% (Figure 7, left and right), corroborating the results shown in Subsection 5.1. The Dim2_Cube algorithm provides gains of up to ~20% (Figure 7, left and right). The TSTS_Cube algorithm achieves a gain of up to ~15% for small networks (Figure 7, left), but it does not provide any gain for large networks (Figure 7, right).

6 Conclusion

All-to-all communication is extensively used by parallel algorithms, and adapting these algorithms to execute efficiently in heterogeneous networks is crucial to improve their performance in this kind of environment.

In this paper, we have presented strategies to organize the nodes in a heterogeneous network into a hypercube. The strategies are based on the communication cost between the nodes in the network. The experiments performed have shown that the Eff_Cube algorithm presented helps to lower communication costs and, consequently, to improve the performance of algorithms based on all-to-all communication.

References

1. M. Banikazemi, V. Moorthy, and D. K. Panda, “Efficient Collective Communication on Heterogeneous Networks of Workstations,” in Proceedings of the ICPP’98, August 1998.
2. M. Banikazemi, et al, “Communication Modeling of Heterogeneous Networks of Workstations for Performance Characterization of Collective Operations,” in Proceedings of the Heterogeneous Computing Workshop, April 1999.
3. M. Bernaschi and G. Iannello, “Collective Communication Operations: Experimental Results vs. Theory,” Concurrency: Practice and Experience, vol. 10, no. 5, pp. 359-386, 1998.
4. D. P. Bertsekas, et al, “Optimal Communication Algorithms for Hypercubes,” Journal of Parallel and Distributed Computing, vol. 11, pp. 263-275, 1991.
5. C. Chen and R. Chen., “Compact Embeddings of Binary Trees into Hypercubes,” Information Processing Letters, vol. 54, no. 2, pp. 69-72, April 1995.
6. S. M. Figueira and C. Mendes, “Dynamically Adaptive Binomial Trees for Broadcasting in Heterogeneous Networks of Workstations,” in Proceedings of the VECPAR, June 2004.
7. I. Foster, “Designing and Building Parallel Programs - Concepts and Tools for Parallel Software Engineering,” Addison Wesley Publishing Company, 1995.
8. I. Foster, et al, “Wide-Area Implementation of the Message Passing Interface,” Parallel Computing, vol. 24, no. 12, pp. 1735-1749, 1998.
9. I. Foster and N. Karonis, “A Grid-Enabled MPI: Message Passing in Heterogeneous Distributed Computing Systems,” Proceedings of the Supercomputing’98, November 1998.
10. M. R. Garey and D. S. Johnson, “Computers and Intractability: A Guide to the Theory of NP-Completeness,” W. H. Freeman, San Francisco, 1979.
11. A. Geist, A. Beguelin, J. Dongarra, R. Manchek, W. Jiang, and V. Sunderam, “PVM: A Users' Guide and Tutorial for Networked Parallel Computing,” MIT Press, 1994.
12. V. Heun and E. Mayr, “Efficient Dynamic Embeddings of Binary Trees into Hypercubes,” Technical Report TR-98-023, International Computer Science Institute, Berkeley, California.
13. S. L. Johnsson, “Communication Efficient Basic Linear Algebra Computations on Hypercube Architectures,” Journal of Parallel and Distributed Computing, vol. 4, no. 2, pp. 133-172, 1987.
14. N. Karonis, et al, “Exploiting Hierarchy in Parallel Computer Networks to Optimize Collective Operation Performance,” Proceedings of the 14th IPDPS, pp. 377-84, May 2000.
15. T. Kielmann, et al, “MagPIe: MPI’s Collective Communication Operations for Clustered Wide Area Systems,” in Proceedings of the PPoPP’99, May 1999.
16. T. Kielmann, H. E. Bal, and S. Gorlatch, “Bandwidth-Efficient Collective Communication for Clustered Wide Area Systems,” in Proceedings of the IPDPS’00, May 2000.
17. F. T. Leighton, “Introduction to Parallel Algorithms and Architectures,” Morgan Kaufmann, 1992.
18. M. Livingston and Q. Stout, “Embeddings in Hypercubes,” Mathematical and Computational Modeling, vol. 11, pp. 222-227, 1988.
19. C. Loan, “Computational Frameworks for the Fast Fourier Transform,” SIAM, 1992.
20. B. Lowekamp and A. Beguelin, “ECO: Efficient Collective Operations for Communication on Heterogeneous Networks,” in Proceedings of the 10th International Parallel Processing Symposium, April 1996.
21. Message-Passing Interface Forum, “MPI: A Message-Passing Interface Standard,” International Journal of Supercomputing Applications, 8(3/4), 1994.
22. MPICH-A Portable Implementation of MPI, http://www-unix.mcs.anl.gov/mpi/mpich/.
23. M. Quinn, “Parallel Computing - Theory and Practice,” McGraw-Hill, 1994.
24. S. Ranka and S. Sahni, “Hypercube Algorithms for Image Processing and Pattern Recognition,” Springer-Verlag, 1990.
25. P. Swarztrauber, “Multiprocessor FFTs,” Parallel Computing, vol. 5, pp. 197-210, 1987.
26. Y. Tseng, et al, “Low-Congestion Embedding of Multiple Graphs in a Hypercube,” International Conference on Parallel and Distributed Systems, pp. 378-385, 1992.
27. R. Wolski, N. Spring,. and J. Hayes, “The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing,” in *Journal of Future Generation Computer Systems*, 1999.