

# ECO: Efficient Collective Operations for Communication on Heterogeneous Networks

Bruce B. Lowekamp\* and Adam Beguelin†

Email: {lowekamp,adamb}@cs.cmu.edu

School of Computer Science

Carnegie Mellon University

Pittsburgh, PA 15213

Phone: 412-268-5295

Fax: 412-268-5576

September 19, 1995

## Abstract

PVM and other distributed computing systems have enabled the use of networks of workstations for parallel computation, but their approach of treating a network as a collection of point-to-point connections does not promote efficient communication—particularly collective communication. ECO is a package which solves this problem with programs which analyze the network and establish efficient communication patterns which are used by a library of collective operations. The analysis is done off-line, so that after paying the one-time cost of analyzing the network, the execution of application programs is not delayed. This paper gives performance results from using ECO to implement the collective communication in CHARMM, a widely used macromolecular dynamics package. ECO facilitates the development of data parallel applications by providing a simple interface to routines which use the available heterogeneous networks efficiently. This approach gives a naive programmer the ability to use the available networks to their full potential without acquiring any knowledge of the network structure.

## 1 Introduction

The availability of networks of high-performance workstations and software packages such as PVM [7] has made networks of workstations (NOWs) a legitimate alternative to traditional high-performance machines such as supercomputers and massively parallel processors

---

\*Partially supported by an NSF Graduate Research Fellowship

†Joint appointment with Pittsburgh Supercomputing Center

(MPPs). Furthermore, networks of supercomputers can be utilized when even more computational power is needed. However, the networks forming NOWs are almost never as powerful as the networks within MPPs, so most applications run on NOWs have been coarse-grain computations which require relatively little communication. The advent of high-performance and gigabit networks, such as 100Mb ethernet [14], FDDI, ATM [16], and HIPPI [9] networks has begun to reduce this limitation, however few users are lucky enough to have any or all of their machines on such networks, therefore it is critical that efficient use is made of what network bandwidth is available. Even with high-performance networks, it is still necessary to use appropriate communication patterns based on the topology of these networks. This is made difficult by the fact that local area networks, unlike MPP networks, tend to hide information about topology.

ECO is focused on the optimization of communication in the data parallel computational model. Most communication in data parallel programs can be divided into two categories: nearest neighbor, where processors communicate to exchange data with a small set of other processors which contain “nearby” data, and collective, or global, communication, where all processors contribute data to a result that may arrive at one or all processors. Optimization of both types of communication should be addressed by a complete communications package.

Collective operations have long been a component of vendor supplied communication libraries for MPPs, and the supplied routines are optimized for performance on that vendor’s hardware. PVM briefly addresses these issues and MPI [8, 6] provides a more complete set of collective communication routines, however neither PVM nor MPI’s specification address the issues of optimizing the performance of collective communication on heterogeneous networks.

Collective communication can be implemented using the native multi-cast capabilities of networks such as ethernet and ATM, however there are several difficulties with using this solution. Such a technique would only be usable on the same local network, requiring alternate solutions for systems spanning multiple networks. There are currently no portable techniques to identify an underlying physical network. Finally, the multicast protocols may be unreliable, which further complicates an implementation of reliable collective communication needed for applications.

ECO addresses these concerns by analyzing the characteristics of the networks to which the machines are attached and developing optimized communication patterns which are used by collective communications routines. These routines provide the same functionality as the collective communication suite in the MPI standard [6]. Efficient nearest neighbor communication is provided by routines that map common communication topologies to the network topology. ECO requires no user input to determine the characteristics of the network and has almost no application run-time overhead. Its design makes it possible to utilize more efficient communication techniques, such as those provided by MPP libraries, while maintaining the flexibility to run on an arbitrary collection of machines and networks.

ECO is used for collective communication by Dome [1], an object-oriented distributed programming environment under development. It has also been used to provide the collective communication required by CHARMM [?], a macromolecular dynamics program extensively used by chemists. The communication routines provided with CHARMM are highly optimized for a switched or high-speed network. ECO provides substantial improvements in the run-time of CHARMM on heterogeneous networks, while suffering only a slight penalty in efficiency on switched networks.

## 2 Related Work

Collective communication provides a set of core routines for many applications. The inclusion of efficient implementations of core collective operations is crucial for achieving maximum performance of applications on message-passing systems [13].

PVM currently lacks a powerful implementation of collective communication. Moreover, such an implementation presents difficulties because of the portable, heterogeneous nature of PVM. While the developer of an MPP library knows the topology and characteristics of the interconnection network, the developer of such a library for PVM has no such assumptions to work with. This does not, however, reduce the need for such a facility in PVM. Developers of applications in PVM have been forced to design their own collective communication support [15]. While coding this directly forces the developers to make decisions specific to

their network, it introduces excessive difficulties.

There has already been substantial work on collective communication packages. The MPI standard [6] acknowledges the importance of collective communication by including it as a chapter in the specification. Papers from the InterCom project discuss general techniques for building high-performance collective communication libraries, implementation of their library on several architectures [3], and discussions of other packages and approaches to collective communication [13].

Bala, et al. describe a collective communication library originally designed for the IBM SP1 [2]. They discuss performance tuning issues, as well as a detailed discussion of the semantics of collective communication and group membership, including the correctness of collective operations.

Considerable work has been done on collective operations, and multicast communication in general, which can be used as the underpinnings for collective communication, on a variety of specific physical networks. McKinley, et al. have written several papers on issues involved in implementing such operations on bus-based networks [12], wormhole routed MPPs [11], and ATM networks [10].

Another feature missing from PVM which application developers must provide [15] is the notion of topology among tasks. Many developers have expressed a desire for a basic set of topologies, such as ring and mesh topologies. The MPI standard dedicates a chapter to the discussion of a mechanism for describing arbitrary topology needs to the message passing system [6].

Of particular interest to ECO development is research done on grouping hosts on the basis of network topology. This technique has been used in two areas. Evans and Butt make use of this technique to facilitate load balancing [5]. In their approach, full load balancing information and communication occurs within subnets, while communication between sub-networks is more carefully controlled. Also related to this subject is the work of Efe on grouping related tasks together in a subnet for a deterministic task system [4]. Both of these systems share ECO's principle of limiting communication between subnets. However, a major difference between these systems and ECO is that they attempt to avoid global

communication whereas ECO tries to optimize it. Furthermore, neither system addresses the issue of automatically identifying subnets.

### 3 Network Characterization

ECO is designed so that the application programmer need know nothing of the underlying network topology. Accordingly, the network characterization program must function starting only with the list of machines to be used. The goal of the characterization program is to obtain a metric which can be used to judge the performance of each link.

The first issue in network characterization is the physical structure of the network. A full description of a network can be very complicated. There are two broad classifications of networks: bus-based and switch-based. In a bus-based network several hosts share the same “wire.” Each host in a switched network has its own wire and a switch connects pairs of wires as they exchange messages. Bridges can be used to connect different network components together, and can directly or indirectly slow down message transfer rates, but are invisible to most software. Routers are used to connect networks and generally delay packets for longer than bridges. MPPs have very fast internal networks, but the connection between the MPP and the outside network, such as on the Cray T3D or Intel Paragon, frequently has relatively low bandwidth.

Secondary to a physical description of the network is the computational load in preparing the message on the sending and receiving processors. The processor has to marshal the data and attach several headers before placing the message on the network. This cost can vary widely depending on the message passing implementation, operating system, and hardware, but is frequently a substantial consideration.

A third issue which affects the communication work is the traffic on the network caused by sources other than the application in question, which is referred to as ambient traffic in this article. This traffic further reduces bandwidth available to an application and increases the latency of message transfers.

Clearly, developing an analytical expression to account for all three factors would be

almost impossible in an automatic system such as that desired in ECO. Therefore, this approach was rejected in favor of one where the communication time between two hosts is measured by timing round-trip messages. This metric is desirable because it expresses the sum of all these terms in a single measurable quantity.

To measure these times, a program is run on all hosts which are anticipated to be used for running parallel programs. A host exchanges a message with another host several times. The total time is divided by twice the number of round trips and recorded. Several round trips are used for each measurement to minimize the influence of the clock granularity. The communication time is measured with one pair of hosts exchanging messages at a single time, in order to prevent the program from causing network congestion which would distort the results. This exchange is repeated for each pair of hosts, and the whole process is repeated several times.

Although this measurement process is an  $O(p^2)$  process when run on  $p$  machines, it does not take excessively long and requires little processor time, which should make it inexpensive for those who pay for CPU time. Typical execution times range from one minute for about 10 hosts to 15 minutes for approximately 50 hosts.

After the measurement is completed, it is necessary to label the communication time for each pair of hosts with a single value. Initially, it seems intuitive that the mean observed time would most accurately reflect the cost to communicate between hosts. However, experience has shown that the mean time is a poor indicator, due to the large delay which can be caused by collisions. The high cost of delays caused by a brief burst of heavy traffic can be made worse by the exponential back-off scheme used on ethernet. A single exchange which experiences this type of effect can skew the mean so that hosts which may share the same network bus appear to have a worse network connection than those hundreds of miles away. Although these collisions result from ambient traffic and should be accounted for, it would be necessary to run the program over a period of several hours to gain even an approximate measure of the frequency of these occurrences.

Other possible measures are the minimum, maximum, and median times. The maximum time will reflect the worst collision which occurred, as discussed above. The median time

is likely not to reflect the delays caused by infrequent collisions or traffic burstiness, if enough measurements are taken. The minimum time has been chosen for use by ECO. This is because, as discussed in Section 4.2, ECO uses these measurements to determine the physical structure of the network, and the minimum time most accurately reflects this, as it reflects the least influence from ambient traffic. The minimum time also has the advantage of allowing the smallest number of measurements to obtain an accurate result.

## 4 Optimizing Collective Communication

There are several important collective operations. To introduce ECO's approach, consider a broadcast, where one processor wants to send a message to all other processors. Our goal is to utilize the time measurements taken previously to determine an efficient communication pattern for broadcasts.

A few terms must be defined first. A processor is referred to as a node. The communication pattern is represented as a directed graph, with edge AB representing a message sent from node A to node B. The depth of a node is the number of edges between it and the broadcasting node. The width of the graph at a given depth is the number of nodes at that depth. The cost of an edge is the measured time for communication between nodes A and B.

The performance of a broadcast is determined by the communication completion time, which refers to the elapsed time between when message is first sent out from the broadcasting node to the time when the message is received by all nodes. Unfortunately, it is not possible to determine this measure exactly using the edge costs. This is because the measurements determine the time from initiating a send to the data being available at the receiving end. However, a host can begin transmitting a second message as soon as the first has reached its network adaptor, which can happen significantly before the message is available at the receiving host. The discussion in this section will assume that it is possible to calculate the communication completion times by adding up the edge costs, but this is done for illustration only and does not work in the general case.

from	M.S.T.	Binary
a	7	9
d	8	7

Table 1: Communication completion times for minimum spanning and binary tree broadcast algorithms on the network shown in Figure 1.

It is immediately evident that the proper representation for a broadcast communication pattern is a tree. To optimize this form of a communication pattern, one intuitive algorithmic approach is to use the minimal spanning tree, which assures the smallest total edge cost.

Consider the network showing in Figure 1a, consisting of six workstations, distributed with two on each of three separate subnets. A minimal spanning tree for this network is shown in Figure 1b. In Table 1, the communication completion time of a broadcast from node a on this tree are compared to a naively formed binary tree, as shown in Figure 1c. This tree was generated by indexing the hosts in alphabetic order, from 1 to 6. Node  $i$ 's children are nodes  $2i$  and  $2i + 1$ . In this case the MST-based pattern performs better than the binary tree.

However, the MST approach does not do well for all cases. Consider the same network, but now with a broadcast from node d. Now, the execution time for the MST-based pattern is inferior to that of the binary tree shown in Figure 1d, which was constructed by ordering the hosts beginning with node d and wrapping around. This shows one of two major failings of the MST approach. The first is that it fails to account for the location of the originating node in determining the pattern, and thus broadcasts may have to follow a wide tree across its entire width, with little or no parallelism. The second, and more serious failing, is that it does not optimize the depth of the tree. This means that for large numbers of processors, the execution time of the graph can be very large.

What is needed is an approach which focuses on reducing communication completion time. We have found no standard graph algorithm which remedies these problems, but a few reasonable assumptions leads to a good heuristic approach.

When looking at the results of the MST approach, as seen in the broadcast from node a

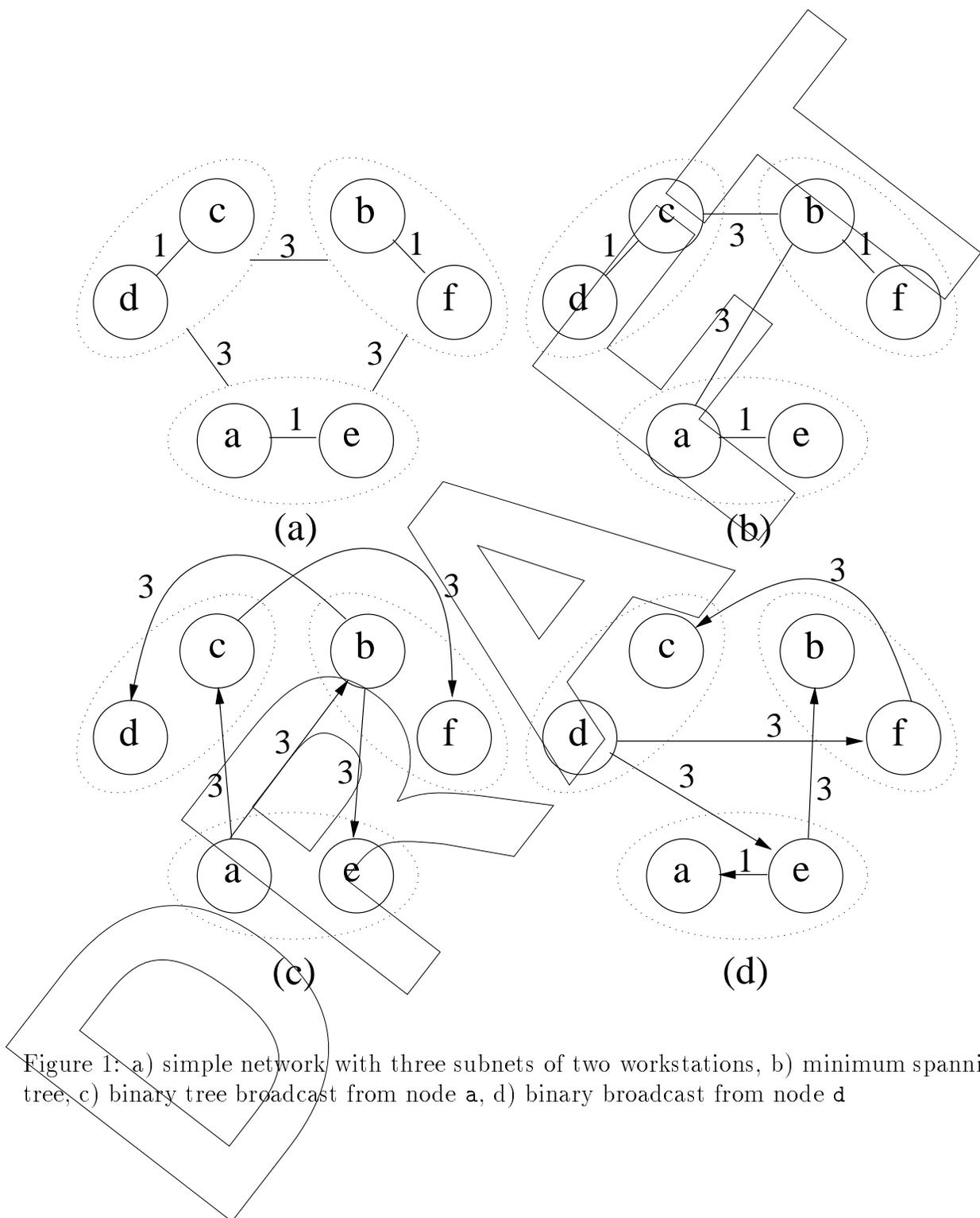


Figure 1: a) simple network with three subnets of two workstations, b) minimum spanning tree, c) binary tree broadcast from node a, d) binary broadcast from node d

on the MST in Figure 1b, several important characteristics appear.

- The overall depth of the tree is small.
- The tree is well balanced with the originator at the root.
- Most communications occur with hosts in the same physical network.

This motivates the following approach:

1. Divide the overall network into “subnets,” which consist of hosts which are in the same physical network. A host is on the same subnet as hosts with which it has its lowest edge costs.
2. Position the originator of the broadcast at the root of the tree.
3. Create a tree using the subnets as vertices rather than the individual processors. Edges on the tree now represent inter-subnet communication.
4. Optimize the intra-subnet communication using patterns appropriate to that network type.

#### **4.1 Partitioning the network into subnets**

The algorithm used to partition hosts into subnets is given in Figure 2. The key criteria for nodes to be neighbors is that the cost of the edge between them be within 20% of the cost of the least expensive edge incident on each of them and within 20% of the least expensive edge within the subnet. Although arbitrary, the 20% cutoff has proven excellent at accurately partitioning networks.

Note that when a node is added to a subnet the algorithm does not check that the cost of all edges from the new node to members of that subnet are within 20% of the new node’s minimum cost edge. This has the advantage that a small number of inaccurate measurements can be made in the network timings without causing incorrect partitioning. This fault tolerance allows for the initial measurements to be taken quickly, since a single

```

initialize subnets to empty

for all nodes
    node.min_edge = minimum cost edge incident on node

sort edges in nondecreasing order

for all edges(a,b)
    if a and b are in the same subnet
        continue
    if edge.weight > 1.20 * node(a).min_edge or
       edge.weight > 1.20 * node(b).min_edge
        continue
    if node(a) in a subnet
        if (edge.weight > 1.20 * node(a).subnet_min_edge
           continue
    if node(b) in a subnet
        if (edge.weight > 1.20 * node(b).subnet_min_edge
           continue
    merge node(a).subnet and node(b).subnet
    set subnet_min_edge to min(edge, node(a).subnet_min_edge,
                               node(b).subnet_min_edge)

```

Figure 2: Algorithm used for partitioning the network into subnets

inaccurate measurement should have few consequences. One could imagine a situation where this approach is incorrect, but we have not encountered such a network in the real world.

The partitioning need only be done once per network. Repartitioning need only be redone when the physical network changes. The results of the partitioning are stored in a file which is loaded when an ECO program is run.

## 4.2 Communication within subnets

Up to this point, all data about ambient traffic has been deliberately avoided. This is because our experiments have shown that the physical network is the most important factor in determining appropriate communication patterns. If a single ethernet, for instance, has a high level of ambient traffic, that would make communication between machines on that ethernet slower, and communication between machines on that ethernet and those not on that ethernet even slower. Therefore, ambient traffic may affect a decision about whether to use machines on a particular network for a computation, but it would not affect decisions about the basic communication pattern, which is to minimize inter-subnet communication.

Ambient traffic is, however, a factor in deciding what communication pattern to use for intra-subnet communication. This decision is influenced by several factors:

- switched or bus-based network,
- bandwidth of the network,
- size of the message, and
- ambient traffic on the network.

Once the network has been partitioned into subnets, it is possible to characterize the network based on the first two points. This should suggest a communication pattern for use on the subnet. Typically parameters generated would be the degree of an appropriate tree, or the maximum depth to which the tree should be expanded, in order to prevent the tree from becoming too wide and generating too much parallel traffic.

Operation	Description
broadcast	one process sends an identical message to all other processes
barrier	all processes reach operation before any complete
scatter	one process sends a different message to each other process
gather	all processes send a message to one or all other processes
all-to-all	all processes perform a scatter and gather
reduce	all processes perform an operation on data, the result is set to one or all processes
scan	each processor receives the results of a reduction on data of processes lower than its rank, inclusive

Table 2: Collective operations supported by ECO

Initial results with this technique have been promising, but it is not presently integrated into ECO. Currently, ECO uses a tree within all subnets, the degree of which can be specified by the user.

### 4.3 Example

Figure 3 shows the application of ECO's technique to a network of machines at Carnegie Mellon University (CMU) and the Pittsburgh Supercomputing Center (PSC). ECO successfully distinguishes between networks with large differences in performance, such as switched FDDI and ethernet, as well as between ethernet networks separated by a bridge. The ring-topology generated by ECO is shown in Figure 3c. The broadcast tree generated by ECO is shown in Figure 3d.

## 5 Collective Operations

Section 4 introduced the techniques that ECO uses with the simple example of a broadcast. As mentioned in Section 1, ECO provides the same functionality as the MPI collective communication standard [6]. Table 2 lists the operations supported by ECO.

The same communication pattern described in Section 4 is used for all operations, with appropriate modifications. For operations involving a single receiver, that receiver is always the root of the tree. Operations such as barrier and gather begin data movement in the

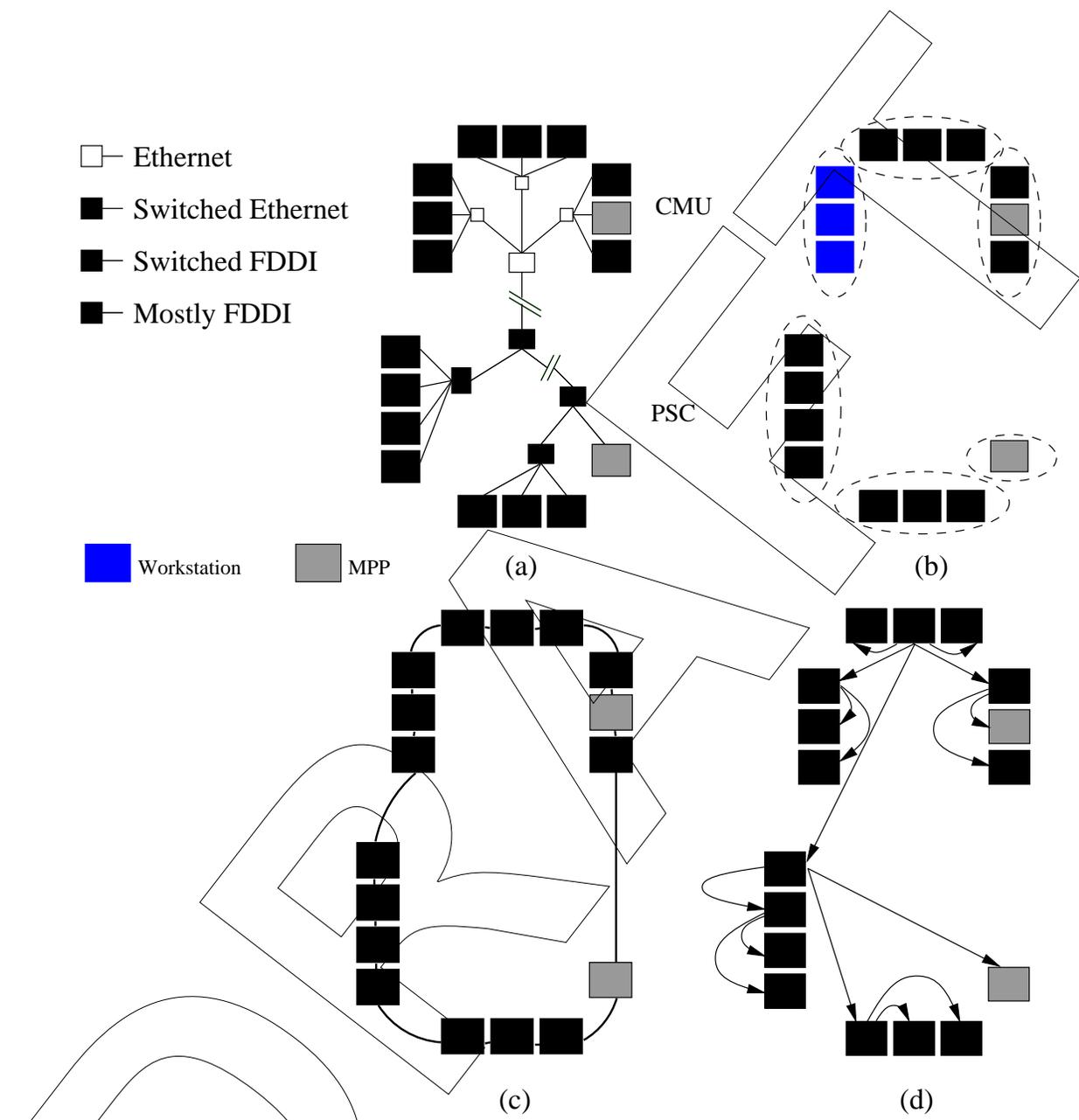


Figure 3: ECO's effect on a heterogeneous network: a) physical network, b) ECO's partitioning into subnets, c) ECO derived ring topology, and d) ECO derived collective communication pattern

leaves of the tree and work toward the root, returning to the leaves in the case of barrier or gather to all. Other operations, such as broadcast, begin at the root and proceed toward the leaves. This pattern is not theoretically optimal, in terms of the number of steps, but experience has shown that on heterogeneous networks it performs better than theoretically optimal algorithms because it makes better use of the available bandwidth.

In order to generate the communication patterns which will be used for the ECO operations, a task must call an initialization procedure. This procedure reads in the subnet data file and organizes appropriate communication patterns for the hosts on which the program is running. Subsequent calls to ECO operations need only use the appropriate communication pattern for the root of the operation, therefore there is little overhead involved.

## 6 Benchmarks

Measuring the performance of a parallel program is notoriously difficult. We use two methods to analyze the effectiveness of ECO's approach. The first is to measure the time of individual operations and the second is to measure the time of execution for an application using ECO.

Four communication patterns were used in these benchmarks: ECO's custom patterns, as well as  $k$ -ary trees ( $k = 2, 3$ ), star, and a butterfly pattern.

### 6.1 Micro-benchmarks

Due to the inherent difficulty in timing a parallel operation, only operations which are started from the root, e.g. broadcast, have been measured directly. This was done by determining the offsets between system clocks among the machines used, recording the time before the operation began and determining the latest time at which the message was received. Clock drift or adjustments can be a factor in such experiments, but no such drift was observed and these machines do not adjust their clocks automatically.

Results are shown in Table 3 for the broadcast of a message of 16000 bytes on eight DEC Alphas, distributed among three 10 Mb ethernet segments joined by a Cisco 7505 bridge. In this case, the butterfly pattern would produce the same results as the binary tree. The

method	mean	median	minimum
ECO	0.119(.008)	0.0651	0.0614
tree( $k = 2$ )	0.174(.007)	0.128	0.104
tree( $k = 3$ )	0.162(.007)	0.122	0.104
star	0.141(.007)	0.109	0.104

Table 3: Time, in seconds, for a 16000 byte broadcast on eight DEC Alphas

pattern	mean	median	minimum
ECO	2.1(.2)	2.17	1.79
tree( $k = 2$ )	2.7(.1)	2.66	2.65
tree( $k = 3$ )	2.3(.1)	2.30	2.12
star	2.7(.2)	2.59	2.47
CHARMM butterfly	3.0(.4)	2.95	2.69

Table 4: Communication time, in minutes, for CHARMM running on a network of eight DEC Alphas

measurement was repeated 1000 times with relatively little ambient traffic.

## 6.2 Application Programs

ECO has been used to provide collective communication for CHARMM [?], a macromolecular dynamics program used by many chemists. CHARMM's implementation of collective communication uses a butterfly pattern, the performance of which has been optimized fairly heavily. It uses in-place data packing and has almost no computational overhead in its collective communication routines. The collective communication used by CHARMM in this benchmark consists of a large number of gather-to-all and reduce-to-all operations, as well as broadcasts.

The same set of eight DEC Alphas used in Section 6.1 was used to run CHARMM. The measurements were taken during periods of low ambient traffic and repeated five times. Table 4 shows the results.

To verify the performance of ECO on an ethernet with high levels of ambient traffic, a limited number of measurements were also taken, as shown in Table 5. It was not considered

pattern	mean
ECO	13.9
tree( $k = 2$ )	18.7
tree( $k = 3$ )	21.6
CHARMM butterfly	16.6

Table 5: Communication time, in minutes, for CHARMM running on a network of eight DEC Alphas with high ambient traffic

pattern	time
ECO( $k = 2$ )	1.08
ECO( $k = 3$ )	3.28
CHARMM butterfly	0.82

Table 6: Communication time, in minutes, for CHARMM on a switched ethernet network of eight SGI INDYs

a reasonable load on the system to make many of these runs, so each point was taken only twice. Statistical results are not presented.

In order to evaluate the overhead of ECO's routines, CHARMM was also run on a set of eight SGI INDYs connected with switched 10 Mb ethernet. ECO was run with  $k = 2$  and  $k = 3$  trees for the single subnet. There was no other traffic on the network. The results in Table 6 indicate that there is a slight overhead inherent in using ECO, as indicated by the results for  $k = 2$ . The results for  $k = 3$  show the importance of matching the appropriate communication pattern to each subnet. Using wider trees is important on bus-based networks, since it reduces the number of communications which are attempted in parallel, but it hurts the performance on a switched network which can handle the bandwidth.

ECO has also been used to implement the collective communication for Dome [1]. A molecular dynamics program written in Dome has been run on a network of 20 machines, consisting of six DEC Alphas attached to two ethernets, five IBM Power PCs attached to an ethernet, two DEC alphas attached to switched FDDI, and seven SGI INDYs attached to switched ethernet. These tests were run using four communication patterns: ECO's optimized pattern, star, tree ( $k = 4$ ), and ring. The times for communication required by

pattern	mean
ECO	63.0
star	153.5
tree( $k = 4$ )	148.1
ring	414.2

Table 7: Communication time, in seconds, for a Dome molecular dynamics application

this application are shown in Table 7. We have not yet run CHARMM on a heterogeneous collection of machines.

Dome is also the only application which makes use of ECO's topology function, using a ring topology for its load-balancing communications. Use of this function prevents the user from having to order the machines by hand to insure quick load-balancing, and reduces the load-balancing communication time by more than half, compared to times for a randomly distributed arrangement of nodes.

## 7 Future Work

ECO is being developed for eventual release as a collective communication library for PVM. Additional features which will be added to ECO before release include:

- automatic characterization of each subnet to determine the appropriate communication pattern for intra-subnet communication,
- support for MPPs using MPP native collective communication calls within the MPP and ECO pattern for external communication,
- support for static groups, and
- additional topology support.

One weakness of using trees as the communication pattern of choice is that it creates a bottleneck at the root of the tree, which may cause scalability problems for moving large amounts of data, particularly in the all-to-all algorithm, where  $p$  processors transmitting  $n$

bytes of data results in  $pn$  bytes of data travelling through the the root node. Eliminating the single root and having all subnets or a fraction of subnets perform the data exchange may help eliminate this bottleneck. This issue requires careful study to provide an efficient technique which can be adapted using only the characterization information available.

ECO currently bases its communication patterns on the static physical structure of a network. However, some communication parameters, such as the root of the pattern for operations which return data to all processors, could be varied at run-time based on the current network traffic patterns. Further research needs to be done on dynamically adapting to network conditions.

Other goals include using the physical networks' native multicast techniques where possible. This should prove particularly interesting on ATM networks.

## 8 Conclusions

ECO's automatic characterization and network partitioning allow an end-user with little or no knowledge of the physical network structure to write programs with efficient collective communications. The characterization is done offline, with almost no execution costs at program run-time.

ECO has been tested with a wide variety of networks, ranging from several shared ethernet segments connected with a bridge to high-performance switched FDDI networks. ECO can distinguish between these networks and derives communication patterns which exploit the underlying topology. Applications such as CHARMM, a widely-used macromolecular dynamics program written with message passing in Fortran, perform markedly better on heterogeneous networks when using ECO than when using other optimized algorithms for collective communication. On a homogeneous network, ECO exhibits only a slight loss in performance compared to a highly-tuned implementation of collective communication for CHARMM.

## 9 Acknowledgements

We would like to express our thanks to Bill Young for providing us with the molecular dynamics program which has been ported to Dome and for bringing CHARMM to us as a possible target application. Multicomputer grant?

## References

- [1] José Nagib Cotrim Árabe, Adam Beguelin, Bruce Lowekamp, Erik Seligman, Michael Starkey, and Peter Stephan. Dome: Parallel programming in a heterogeneous multi-user environment. Technical Report CMU-CS-95-137, Carnegie Mellon University, April 1995.
- [2] Vasanth Bala, Jehoshua Bruck, Robert Cypher, Pablo Elustondo, Alex Ho, Ching-Tien Ho, Shlomo Kipnis, and Marc Snir. CCL: A portable and tunable collective communication library for scalable parallel computers. In *Proceedings of 8th International Parallel Processing Symposium*, pages 835–844. IEEE Comput. Soc. Press, 1994.
- [3] Mike Barnett, Satya Gupta, David G. Payne, Lance Shuler, Robert van de Geijn, and Jerrell Watts. Building a high-performance collective communication library. In *Proceedings of IEEE Scalable High-Performance Computing*, pages 835–834. IEEE Comput. Soc. Press, 1994.
- [4] K. Efe. Heuristic models of task assignment scheduling in distributed systems. *IEEE Computer*, 19(8):897–916, 1982.
- [5] D.J. Evans and W.U.N. Butt. Load balancing with network partitioning using host groups. *Parallel Computing*, 20:325–345, 1994.
- [6] Message Passing Interface Forum. MPI: A message-passing interface standard. *International Journal of Supercomputer Applications*, 8(3/4), 1994. <http://www.mcs.anl.gov/mpi/index.html>.

- [7] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam. *PVM: Parallel Virtual Machine — A Users' Guide and Tutorial for Networked Parallel Computing*. MIT Press, 1994.
- [8] W. Gropp, E. Lusk, and A. Skjellum. *Using MPI*. MIT Press, 1994.
- [9] Ken Hardwick. HIPPI world—the switch is the network. In *COMPCON Spring 1992*, pages 234–238. IEEE Comput. Soc. Press, February 1992.
- [10] Chengchang Huang and Philip K. McKinley. Design and implementation of global reduction operations across atm networks. In *Proceedings of 3rd IEEE International Symposium on High Performance Distributed Computing*, pages 43–50. IEEE Comput. Soc. Press, 1994.
- [11] Philip K. McKinley, Yih jia Tsai, and David F. Robinson. A survey of collective communication in wormhole-routed massively parallel computers. Technical Report MSU-CPS-94-35, Michigan State University, June 1994.
- [12] Philip K. McKinley and Jane W. S. Liu. Multicast tree construction in bus-based networks. *Communications of the ACM*, 33(1):29–41, January 1990.
- [13] Prasenjit Mitra, David G. Payne, Lance Shuler, Robert van de Geijn, and Jerrell Watts. Fast collective communication libraries, please. Technical Report TR-95-22, The University of Texas, June 1995.
- [14] R.A. Quinnell. Emerging 100-Mbit ethernet standards ease system bottlenecks. *EDN (European edition)*, 39(1):35–36,40, January 1994.
- [15] Andreas Stathopoulos, Anders Ynnerman, and Charlotte F. Fischer. A PVM implementation of the MCHF atomic structure package. *to appear in the International Journal of Supercomputer Applications and High Performance Computing*, 1995. <http://www.vuse.vanderbilt.edu/~andreas/publications/jsa.ps>.

- [16] Ronald J. Vetter. ATM concepts, architectures, and protocols. *Communications of the ACM*, 38(2):30–38, February 1995.

DRAFT