

Characterization of Communication Patterns in Message-Passing Parallel Scientific Application Programs

Technical Report No: HPPC-97-10

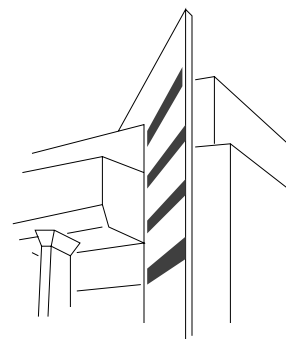
October 1997

JunSeong Kim
David J. Lilja



UNIVERSITY OF MINNESOTA
High-Performance Parallel Computing Research Group

Department of Electrical Engineering • Department of Computer Science • Minneapolis • Minnesota • 55455 • USA



Characterization of Communication Patterns in Message-Passing Parallel Scientific Application Programs

JunSeong Kim David J. Lilja
jskim@ece.umn.edu lilja@ece.umn.edu
Department of Electrical and Computer Engineering
Minnesota Supercomputing Institute
University of Minnesota
200 Union St. SE
Minneapolis, MN 55455

Abstract

This paper examines the communication patterns of parallel scientific programs, including some of the NAS benchmarks and the Miami Isopycnic Coordinate Ocean Model (MICOM), that use explicit message-passing. *Communication locality*, such as communication event locality, message destination locality, and message size locality, is proposed and studied in addition to the widely accepted metrics of message size, destination, and generation distributions. The locality metrics are relatively insensitive to system and problem size variations making them robust metrics for characterizing the communication patterns of parallel applications. The data from this study will be useful for understanding parallel applications' communication behavior and for designing more realistic synthetic benchmarks. We observe that the communication patterns of the benchmark programs are consistent with those of the actual application.

Keywords: message-passing, network-based computing, communication characteristics, communication locality, application-level performance

1 Introduction

Scalable parallel systems have been built using chip-level integration through to system-level integration. System-level integration has been most popular favored in the last few years because of its cost, performance, and flexibility advantages. For instance, parallel processing has been facilitated by the wide-spread use of distributed computing in which a set of independent computers communicate over a network to solve a single large problem. Clusters of workstations have become especially popular for both scientific computing and general-purpose applications. The driving forces behind this trend are the emergence of new faster communication networks, the availability of portable robust communication software, and high-performance microprocessors. In cluster computing environments, communication is a significant performance factor when executing an application in parallel since the communication overhead makes only coarse-grained parallelism feasible, which thereby limits the number of applications that can be usefully parallelized.

A proper understanding of the communication patterns of parallel applications is important for

determining how to maximize their performance within a given environment, and for designing better architectures in the future. While computer network performance evaluation has been a widely researched topic [2, 3, 6, 8, 9], communication performance still is not well understood for scientific computing applications in network-based computing environments. Consequently, it is desirable to verify the fidelity of typical assumptions about communication patterns [2, 8, 9], such as the assumption that message generation follows a Poisson distribution or that message size is exponentially distributed. In addition to leading to a greater understanding of communication behavior, the resulting model will help us in designing more realistic synthetic benchmarks.

This paper quantifies the communication patterns of a wide range of scientific applications. These patterns are characterized using the traditional message destination, size, and frequency distributions, but we also extend this characterization to *message locality*. Specifically, we extend the concept of memory access locality based on the Least Recently Used (LRU) stack model [4, 8] to determine the locality of message destinations, sizes, and consecutive runs of send and receive operations. We also examine the impact of different problem sizes and different numbers of processors on these metrics, and the impact of communication overhead relative to computation time when using several different types of communication networks.

In the remainder of the paper, Section 2 describes the system configuration and benchmarks used in the measurements. Section 3 then presents the measurement methodology, while Section 4 characterizes the communication patterns of the test programs. Finally, Section 5 summarizes our results and conclusions.

2 System Configuration and Benchmarks

2.1 System Configuration

A distributed cluster of four Silicon Graphics Challenge Series servers were used to measure the communication traffic in this study. One node in this system contains eight R10000 processors with the other three nodes each containing four R10000 processors. The processors run at 196 MHz and communicate via a shared bus within a node. All nodes run version 6.2 of the IRIX operating system. As shown in Figure 1, the nodes can communicate with each other via three different physical networks—Ethernet, Fibre Channel, and HiPPI.

The Ethernet is a 10 Mbps bus-like network with distributed access control. It is a contention bus using Carrier Sense Multiple Access with Collision Detect (CSMA/CD) technology. Fibre Channel (FC) [12] is a high-performance serial link supporting various topologies, such as point-to-point, loop, or switch. We use Ancor FCS 250 VME/64 adapters within each node. The nodes are networked together with an Ancor CXT 250 16 Fiber Channel Port Switch running at 266 Mbps. Each machine in the cluster is also equipped with an SGI HiPPI adapter that connects directly to a NetStar HiPPI Switch running at 800 Mbps. HiPPI [5, 12] is a connection-oriented, circuit-switched transport network linked by two 50-pair copper cables.

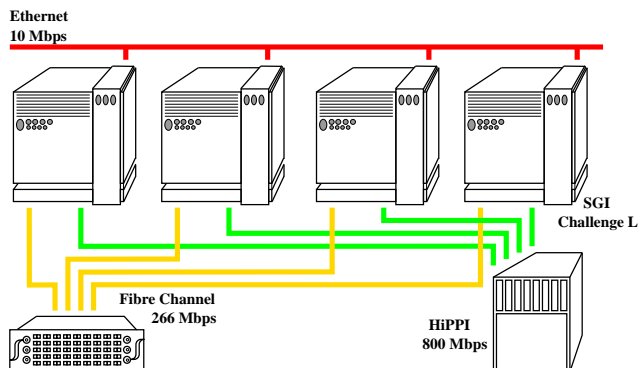


Figure 1: System configuration used in the experiments.

2.2 Programs Tested

We evaluate the communication patterns of several different types of parallel scientific application programs as summarized in Table 1. The *CG*, *MG*, and *IS* programs from the Numerical Aerodynamic Simulation (NAS) parallel benchmarks suite [1], plus six other benchmarks, *Filter*, *Gauss*, *Hough*, *Kirsch*, *TRFD*, and *Warp* [14], are used as examples of kernel benchmarks. We also evaluate the *BT*, *LU*, and *SP* computational fluid dynamics (CFD) applications from the NAS suite. Finally, we include the *Miami Isopycnic Coordinate Ocean Model (MICOM)* [13] program as an example of a large, complete application program. The 127x127 problem size with *MICOM* version 2.6 is run for one simulated day. Any consistency in communication behavior between *MICOM* and the other benchmarks will give us some indication of how well the benchmark programs simulate the communication patterns of a real application program. While any experimental study is limited to drawing specific conclusions for only those applications actually measured, it is felt that these test programs are representative of the types of parallel program kernels one might expect to find running on a typical networked parallel processing system. Figure 3 shows the communication intensity of the test programs in terms of the number of send events per processor and the average number of bytes transferred per send event.

3 Experimental Methodology

All of the application programs have been parallelized using either the Parallel Virtual Machine (PVM) [7] or Message Passing Interface (MPI) [11] message-passing libraries, as shown in the last column of Table 1. Both PVM and MPI are integrated sets of software tools and libraries that provide user-transparent ways to handle message routing across a network. PVM is the existing de facto standard, which was developed as part of a research project on heterogeneous distributed computing. It is built around the concept of a *virtual machine* which is a set of heterogeneous hosts connected by a network that appears to the user as a single large parallel computer. On the other hand, MPI is intended to be a standard message-passing specification that vendors would implement on their systems. It focuses more on performance in message passing and less on the portability of PVM.

The message-passing programming used in these applications is based on just two primitives- SEND

Programs	Description	Parallel Lib.
CG	Uses a conjugate gradient method to compute an approximation of the smallest eigenvalue of a large, sparse, symmetric, positive-definite matrix.	PVM
MG	Solves a 3-D Poisson partial differential equation. This program is a simplified multigrid kernel with constant coefficients, and is a good test for both short- and long-distance data communication.	PVM
IS	Tests a sorting operation that is important in particle method codes. In this benchmark, no floating point arithmetic is involved, but significant data communication is required.	PVM
Filter	Smoothing (averaging) filter that calculates the value of an input image element as the weighted sum of up to 36 neighboring pixel elements in the input image.	MPI
Gauss	Gaussian elimination with back substitution. This program is very communication intensive, requiring several point-to-point, broadcast, and reduction operations.	PVM
Hough	Detects straight lines in the input image, according to the general Hough transform, by finding points of intersections between lines. This algorithm uses both nearest neighbor and global communications.	MPI
Kirsch	Calculates magnitude and direction gradients of an input image.	MPI
TRFD	Simulates a two-electron integral transformation using a fourth-order tensor equation. This algorithm is a series of matrix multiplications and transpositions requiring several point-to-point communication operations.	PVM
Warp	Spatial domain image restoration algorithm that aligns an input image along a given axis. This program exhibits very irregular communications.	MPI
BT	Simulated CFD application. The block tridiagonal benchmark solves multiple independent systems of non-diagonally dominant, block tridiagonal equations with a 5x5 block size.	PVM
LU	Simulated CFD application. The lower-upper diagonal benchmark employs a symmetric successive over-relaxation numerical scheme to solve a regular, sparse, block 5x5 lower and upper triangular system.	PVM
SP	Simulated CFD application. The scalar pentadiagonal benchmark solves multiple independent systems of non-diagonally dominant, scalar pentadiagonal equations with a 5x5 block size.	PVM
MICOM	Miami isopycnic coordinate ocean model [13]. A highly parallel ocean circulation application code characterized by hydrostatic balance in the vertical dimension and a near-geostrophic equilibrium between pressure and Coriolis forces in the horizontal dimension.	MPI

Table 1: Parallel benchmark programs tested.

and RECEIVE. SEND transmits a message from one process to another while RECEIVE reads a message from another process. In PVM, sending a message consists of three steps; 1) Initialization of a send buffer using the `pvm_initsend()` routine, 2) packing of a message into the send buffer using the `pvm_pk*()` routine, and 3) sending the message to another process using the `pvm_send()` routine. A message is received by calling the RECEIVE routine and then unpacking each of the packed items from the receive buffer by calling the `pvm_recv()` and the `pvm_upk*()` routines respectively. Similarly, MPI has the `MPI_SEND()` and `MPI_RECV()` routines but no packing or unpacking procedures since it is used for homogeneous systems. The SEND routines in this experiment are always asynchronous so that computation on the sending process resumes as soon as the message is written to the send buffer. The RECEIVE routine, on the other hand, blocks the receiving process until a message has arrived.

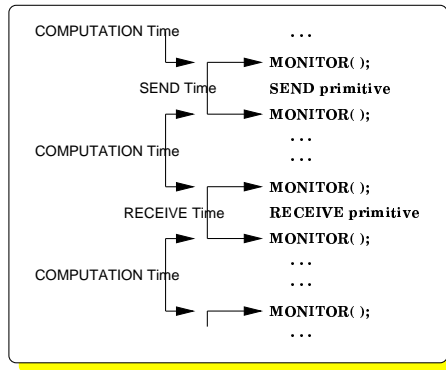


Figure 2: Monitor operations added to the parallel test programs.

To measure the communication patterns, we extend the parallelized versions of the test programs by inserting monitor operations at points in the programs where message-related activities occur. We insert our MONITOR operations before and after the SEND and RECEIVE primitives in the source code, as shown in Figure 2. The MONITOR operation consists of some arithmetic operations to calculate the appropriate characterization metrics. In addition, we insert time-stamps before and after each MONITOR operation to allow us to exclude the monitoring overhead from the execution time measurements. In this way, the execution time of each parallel program is divided into send time, receive time, and computation time. The send time is only the time required to execute the SEND instruction and to move data into the sending buffer on the sending process, since we use an asynchronous send routine. The receive time, however, includes both the receiving overhead and the time the receiver is blocked waiting for a message. The send and receive time can vary since three different types of networks can be used. The computation time will be constant, however, since the processors remain the same.

4 Communication Patterns and Characteristics

This section presents the observation of communication patterns with the test programs. This includes message size and destination distribution, communication localities, effect of problem size and system size on those metrics, and performance effect of different networks.

4.1 Traditional Message Size and Destination Distributions

We call an application *communication intensive* when it transfers messages frequently or when it transfers large amounts of data between processors. Figure 3 shows the communication intensity of the test programs in terms of the number of send events per processor and the average number of bytes transferred per send event. A program that is communication intensive in terms of volume is not necessarily communication intensive in terms of frequency, and vice versa. For instance, *Warp* and *Hough* send the largest messages, but they have relatively few total communication events. On the other hand, the *IS*, *MG*, and *CG* benchmarks are communication intensive in terms of both frequency and volume.

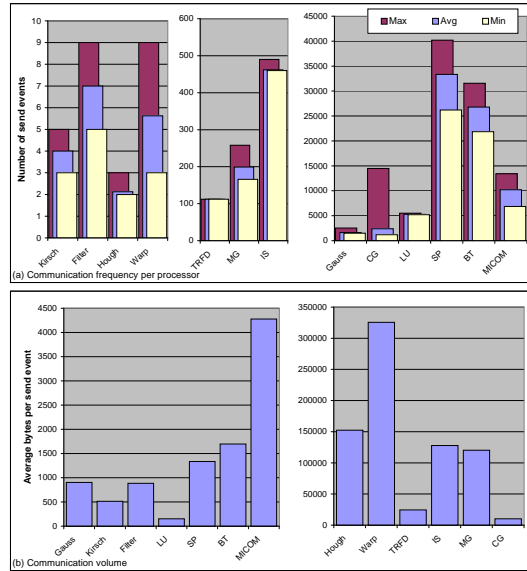


Figure 3: Communication intensity in terms of communication frequency and volume.

Traditionally, the communication requirements of a parallel application have been characterized by three attributes- the *temporal*, *spatial*, and *volume* components [2]. Spatial behavior is characterized by the distribution of message destinations. The typical assumption is that the destinations of messages are evenly distributed among all of the processors. Figure 4 (a) appears to verify that assumption for our test programs since the overall destinations are uniformly distributed among all of the processors. However, destinations are not uniformly distributed from the viewpoint of the individual processors. Figure 4 (b) shows the message destinations of each individual processor for the *CG* benchmark. Except for P0, which is the favorite destination of all of the processors, each processor prefers one to five distinct processors out of the 15 available destinations as their communication partners. The other programs also show favored communication partners except for the *IS*, *TRFD*, and *Gauss* programs, which communicate equally among all processors. We find that this pattern is consistent within an application when varying both the number of processors used to execute the application and the problem size.

The volume of data transferred is characterized by the distribution of message sizes and the average number of messages. In this section, we focus only on the size of the messages transferred, measured in bytes and ignoring header information, since the number of messages is dependent on the number

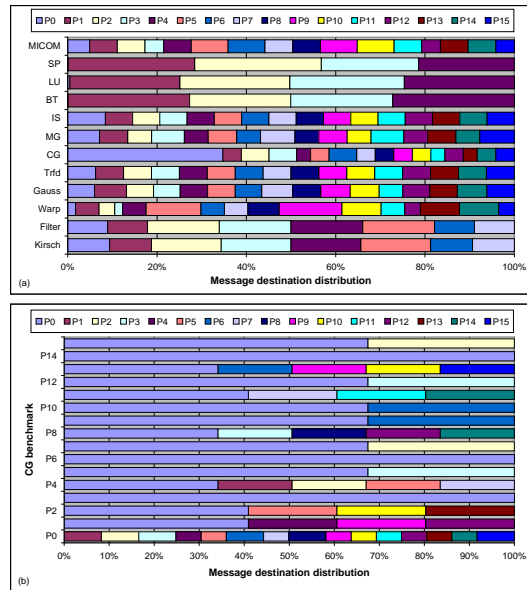


Figure 4: Overall message destination distribution for the test programs and the message destination distribution for each processor in the CG benchmark.

of processors. For message passing primitives, some amount of overhead is unavoidable, since the data transferred must be combined with an appropriate header. This header contains source and destination information, a sequence number, an acknowledgment field, for reliable communication, and so on. In this study, we do not consider the size of any header information focusing instead on the amount of data sent in each message. Figure 5 shows the distribution of message sizes in the test programs, giving a more detailed view of the volume behavior of the applications than in Figure 3. The horizontal axis in this figure represents the message size plotted on a logarithmic scale with the vertical axis showing the cumulative distribution of message sizes. Most benchmarks show two or three clear steps in the distribution indicating that all of the messages within an application have only two or three distinct sizes. Furthermore, some fraction of all the messages within an application tend to be very large while the remainder tend to be very small. For example, about 64% of the messages in the *CG* program are 8 bytes in length whereas the remainder are around 28K bytes. Similarly, in the *IS* program, about 33% are 4 bytes and the remainder are around 130K bytes. Thus, the distribution of the sizes of the messages sent by these applications is bimodal with two widely-separated peaks.

The temporal behavior of the applications' messages is characterized by the distribution of the message generation rate. Figure 6 shows cumulative distribution functions (cdf) of the computation time, the send time, and the receive time of the *CG* and *Gauss* benchmarks when they are executed using different networks. As is often assumed in analytic studies of network performance, the temporal behavior of message-sending appears to be exponentially distributed [2, 9]. There is little difference in the temporal behavior when using Fibre Channel or HiPPI, while the Ethernet shows more delay for send and receive operations. This difference occurs because the Ethernet uses a contention bus while the other two networks use a switched topology.

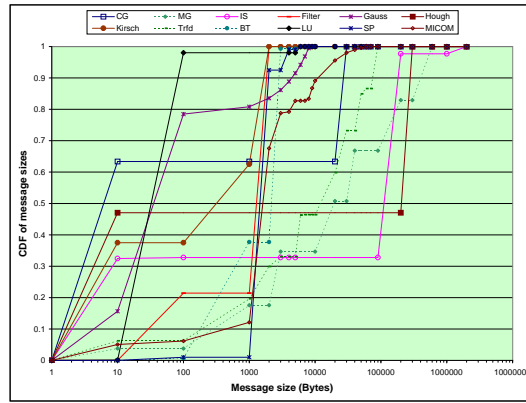


Figure 5: The cumulative distribution of message sizes in the test programs.

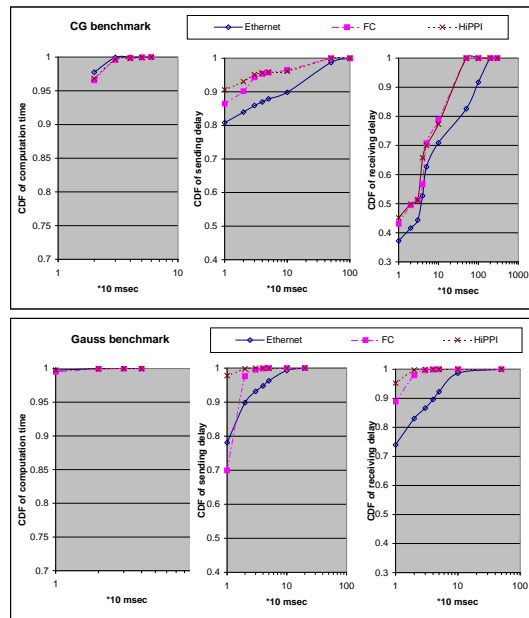


Figure 6: The cumulative distribution function of the computation and communication times.

4.2 Communication Localities

While these measurements tend to confirm both previous studies and intuition of communication characteristics, we extend this standard characterization to investigate the *localities* of communication behavior. Specifically, we quantify patterns and repetitions in types of communication events (i.e. send or receive), message size, and message destination using the Least-Recently Used (LRU) stack model of memory access locality. For example, this model assumes a stack window of size n for each node that contains the processor number of the n most recent message destinations or the size, in bytes, of the n most recent messages sent. If the next message’s destination (or size) is in the window stack, we count a hit. Otherwise, we count a miss. We then define the *locality* of the particular type of event to be $\frac{\# \text{ hits}}{\# \text{ hits} + \# \text{ misses}}$. The window size for the locality of message destinations varies from one to the total number of processors. Various window sizes are used to examine the locality of message sizes.

The *locality of communication events* shows the repetition of send and receive events. For example, completely alternating send and receive events will produce zero locality with a window size of 1. It is interesting to see in Figure 7 that many applications have high (greater than 80%) communication event locality. This means that 80% of the time a send operation is followed by another send operation, or a receive is followed by another receive. We notice in the programs’ source code that there are often communication phases alternating with computation phases. During the communication phase, a bundle of send or receive operations are executed consecutively. This pattern may be due to the nature of the application itself or due to the preference of the programmer. However, this pattern can have a large impact on performance, since overlapping send and receive operations becomes difficult.

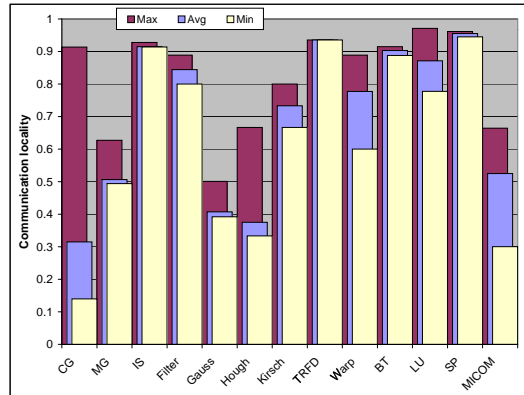


Figure 7: Communication event (i.e. send/receive) locality.

While Figure 4 showed that the overall distribution of message destinations is uniform, the *message-destination locality* measurement in Figure 8 shows that in most of the applications, processors have only a small number of favored communication partners. For example, the line representing the MICOM program shows a 1% hit ratio when the window size is 1, meaning that 1% of the time, when it sends a message to one processor, its next message is sent to that same processor. When the window size is expanded to 2, however, we find that 25% of the time when a processor sends message, it is sent to one of the two processors to which it has most recently sent messages. Continuing to expand the window size, we find that nearly 99% of the time, this program sends its messages to one of the four processors to which it has most recently sent messages.

Figure 8 shows that this type of communication locality is common among the applications tested. Although it is not shown in these figures, we find that the processors tend to communicate with relatively disjoint sets of destination processors. As a result, even though individual processor communicate with only a small subset of those in the system, the overall distribution of message destinations tends to appear uniformly distributed.

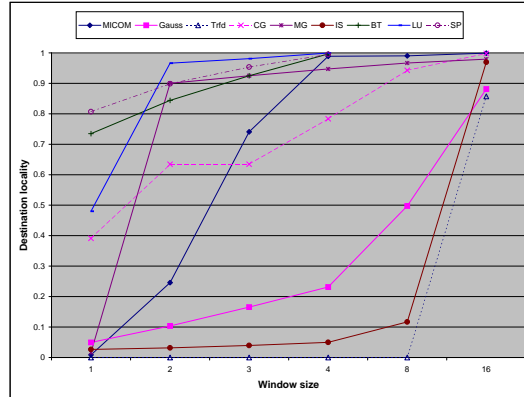


Figure 8: Message destination locality.

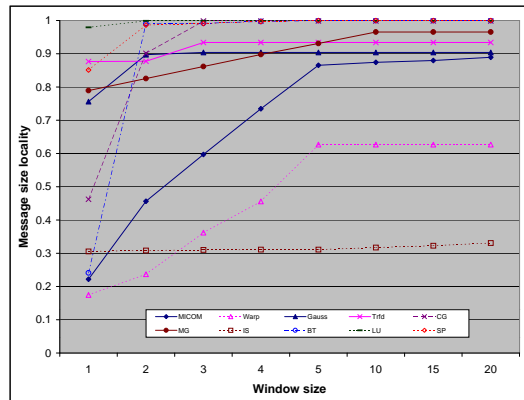


Figure 9: Message size locality.

Figure 9 shows *message size locality*, extending the message size distribution results in Figure 5 to include a temporal component. It is clear from Figure 9 that, since the message size locality of most programs with window size two or three is more than 90%, there are just two or three distinct message sizes used in applications. The exception is the *IS* benchmark which shows message size locality of only around 33%. However, as shown in Figure 5, this benchmark has 67% of its messages clustered around 130K bytes with only small differences. Since we count only messages with exactly the same size as LRU hits, this program shows poor locality. If we counted messages sizes that were *close* in size as hits, though, this program would show similar behavior as the others.

4.3 Effect of Problem Size and System Size Variations

We next examined the impact of varying the problem size and the number of processors on the communication characteristics of the test programs. Two to sixteen processors and three different problem sizes were considered for most of the test programs. In general, when additional processors are used to execute an application, the communication frequency tends to increase while the average size of each message decreases. These changes occur since, as more processors share the same data set, each processor handles a smaller fraction of the total data. Also, when the problem size increases, the communication frequency may either increase or remain roughly the same. However, the average size of each message increases since there is more total data. Thus, larger problem sizes produce *larger* messages instead of *more* messages.

While the size and number of messages varies with the problem size and the number of processors, the localities are quite unaffected. As shown in Figure 10, only communication event locality increases as the number of processors increases. That is, any change in the number of processors produces a change in communication frequency. As the communication frequency increases, there are longer runs of consecutive send or receive events during a given communication phase. This change in communication behavior is shown by the increase in communication event locality as the number of processors, n , increases in Figure 10. Since the other locality parameters are relatively insensitive to changes in the number of processors or the problem size, they should be very useful to characterize an application's communication patterns.

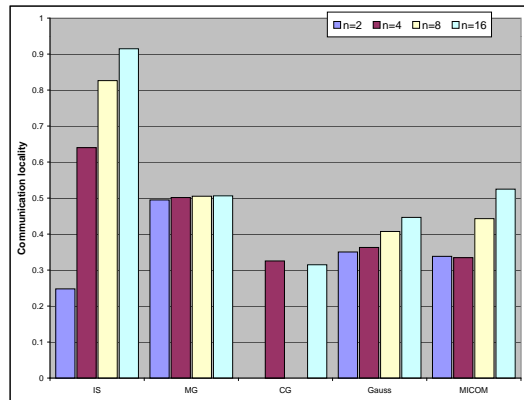


Figure 10: Changes in communication event (i.e. send/receive) locality as the number of processors is changed.

4.4 Performance Effect of Network Type

Another interesting issue is the application-level performance of these benchmarks. Figure 11 shows the execution time of the programs divided into send, receive, and computation time when they are executed using the Ethernet, Fibre Channel, and HiPPI networks. It is surprising to find that communication time dominates the computation time for most of these applications to such a large degree. For many of the applications, the fraction of communication time is significantly larger than the computation time

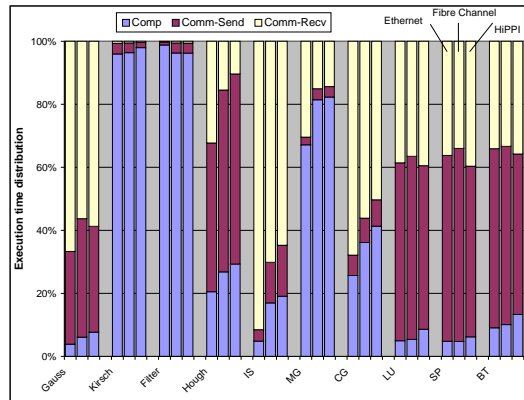


Figure 11: Execution time distributions for the different communication networks.

so that it would be impossible to completely overlap communication time with computation time. As a result, communication time will continue to be a bottleneck in the performance of these applications. To overcome this high communication overhead, more fundamental methods are needed to directly reduce the communication overhead, such as using low-overhead communication protocols or using multiple networks simultaneously [5, 9, 10]. Note that there is little difference in application-level performance when using the different networks even though their differences in peak bandwidth is quite large.

5 Summary and Conclusion

This study has evaluated the communication characteristics of several scientific application programs that have been explicitly parallelized using the PVM and MPI message-passing primitives. We have studied these applications on a cluster of Silicon Graphics multiprocessor workstations interconnected with Ethernet, HiPPI, and Fibre Channel communication networks. Our results have confirmed earlier studies [2, 3, 8, 9] that have used traditional communication characterization metrics. Specifically, we found that the overall distribution of message destinations appears to be uniform, and that the distribution of the size of messages tends to be bimodal. That is, the messages sent by the processors executing the application tend to be either very large or very small, and each processor has an equal likelihood of being the destination of a send operation.

To dig deeper into the characteristics of the communication behavior of these parallel application programs, however, we extended these traditional metrics to incorporate the concept of the *locality of communication events*. Based on a technique derived from the LRU stack model of locality, we come to the following conclusions.

- Send and receive operations typically occur in runs of consecutive sends or runs of consecutive receives. In fact, approximately 80% of the time, a send operation is followed by another send, or a receive is followed by another receive.
- Using the *destination locality* metric, we found that, instead of being uniformly distributed as previously assumed, each processor tends to have only a small number of favored destination

processors for the messages it sends. These destinations tend to be relatively disjoint, however, so that the overall *cumulative* distribution of message destinations does tend to appear uniform.

- The *message size locality* metric confirms that most application programs have only two or three distinct message sizes that processors send.
- While the *frequency* of communication events tends to increase as the number of processors used to execute an application increases, and the sizes of the messages tend to increase as the problem size increases, the values of the locality metrics are relatively insensitive to changes in both the number of processors and the problem size.

We also found that, from the application performance point-of-view, the communication time severely dominates the computation time. This characteristic holds true even as the peak bandwidth of the communication network is dramatically increased. Even if the communication time of the applications could be completely overlapped with the available computation time, there would still be “leftover” communication time that could not be hidden by the computation time. Thus, this communication time is the major performance bottleneck.

Based on these measurements, we conclude that the proposed locality metrics could be very useful in characterizing the communication behavior of message-passing parallel application programs. We further conclude that, even with significant increases in raw network bandwidth, communication delays continue to be an important limitation in using networked clusters of workstations to speed-up the execution of scientific application programs.

Acknowledgments

We thank Steven VanderWiel and Aaron Sawdey for all their help with the test programs. This work was supported in part by National Science Foundation grant no.CDA-9414015.

References

- [1] D. Bailey, et al., “The NAS Parallel Benchmarks,” *NAS Report RNR-94-007*, March 1994.
- [2] S. Chodnekhar, et al., “Towards a Communication Characterization Methodology for Parallel Applications,” *High-Performance Computer Architecture*, 1997.
- [3] R. Cypher, A. Ho, S. Konstantinidou, and P. Messina, “Architectural Requirements of Parallel Scientific Applications with Explicit Communication,” *International Symposium on Computer Architecture*, 1993, pp. 2-13.
- [4] J. L. Hennessy, and D. A. Patterson, *Computer Architecture A Quantitative Approach*, MorganKaufmann Publishers, 1996.
- [5] J. Hsieh, D. H. C. Du, N. J. Troullier, and M. Lin, “Enhanced PVM Communications over a HIPPI Networks,” *Proceedings of the Second International Workshop on High-Speed Network Computing*, April 1996.
- [6] R. Fatoohi, and S. Weeratunga, “Performance Evaluation of Three Distributed Computing Environments for Scientific Applications,” *Supercomputing'94*, 1994, pp. 400-409.
- [7] A. Geist, et al., *PVM: Parallel Virtual Machine - A Users' Guide and Tutorial for Networked Parallel Computing*, The MIT Press, 1994.
- [8] J.-M. Hsu, and P. Banerjee, “Performance Measurement and Trace Driven Simulation of Parallel CAD and Numeric Applications on a Hypercube Multicomputer,” *International Symposium on Computer Architecture*, 1990, pp. 260-269.
- [9] J. Kim, and D. J. Lilja, “Exploiting Multiple Heterogeneous Networks to Reduce Communication Costs in Parallel Programs,” *Heterogeneous Computing Workshop, International Parallel Processing Symposium*, April 1997, pp. 83-95.

- [10] J. Kim, and D. J. Lilja, "Utilizing Heterogeneous Networks in Distributed Parallel Computing Systems," *International Symposium on High Performance Distributed Computing*, August 1997, pp. 336-345.
- [11] Message Passing Interface Forum, *MPI: A Message-Passing Interface Standard*, Version 1.1, June 1995.
- [12] S. Saunders, *The McGraw-Hill High-Speed LANs Handbook*, McGraw-Hill, 1996.
- [13] A. Sawdey, "Using the Parallel MICOM on SGI Multiprocessors and the Cray T3D," *The MICOM User's Group Meeting*, February 1995.
- [14] S. VanderWiel, D. Nathanson, and D. J. Lilja, "Complexity and Performance in Parallel Programming Languages," *International Workshop on High-Level Parallel Programming Models and Supportive Environments, International Parallel Processing Symposium*, April 1997, pp. 3-12.