Communication Characteristics in the NAS Parallel Benchmarks

Ahmad Faraj Xin Yuan
Department of Computer Science, Florida State University, Tallahassee, FL 32306
{faraj, xyuan}@cs.fsu.edu

***Abstract*—**
In this paper, we investigate the communication characteristics of the Message Passing Interface (MPI) implementation of the NAS parallel benchmarks and study the effectiveness of *compiled communication* for MPI programs. Compiled communication is a technique that utilizes the compiler knowledge of both the application communication requirement and the underlying network architecture to significantly optimize the performance of communications whose information can be determined at compile time (static communications). The results indicate that compiled communication can be applied to a large portion of the communications in the benchmarks. In particular, the majority of collective communications are static.

Keywords: NAS parallel benchmarks, Communication Characteristics, Compiled Communication.

I. Introduction

As microprocessors become more and more powerful, clusters of workstations have become one of the most common high performance computing environments. The standardization of the Message Passing Interface (MPI) [1] facilitates the development of scientific applications for clusters of workstations using explicit message passing as the programming paradigm and has resulted in a large number of applications being developed using MPI. Designing an efficient cluster of workstations requires the MPI library to be optimized for the underlying network architecture and the application workload.

Compiled communication has recently been proposed to improve the performance for MPI routines for clusters of workstations[9]. In compiled communication, the compiler determines the communication requirement in a program and manages network resources, such as multicast groups and buffer memory, statically using the knowledge of both the underlying network architecture and the application communication requirement. Compiled communication offers many advantages over the traditional communication method. First, by managing network resources at compile time, some runtime communication overheads such as the group management can be eliminated. Second, compiled communication can use *long–lived* connections for communications and amortize the startup overhead over a number of messages. Third, compiled communication can improve network resource utilization by using off–line resource management algorithms. Last but not the least, compiled communication can optimize arbitrary communication patterns as long as the communication information can be determined at compile time. The limitation of compiled communication is that it cannot apply to communications whose information cannot be determined at compile time. Thus, the effectiveness of compiled communication depends on whether the communication information can be obtained at compile time.

While the existing study [4] shows that the majority of communications in scientific programs are static, that is, the communication information can be determined at compile time, the communication characteristics of MPI programs have not been investigated in this manner. Given the popularity of MPI, it is important to study the communications in MPI programs and determine the effectiveness of compiled communication for such programs. In this paper, we investigate the communication characteristics in the NAS parallel benchmarks [2], the popular MPI benchmarks that are widely used in industry and academia to evaluate high performance computing systems.

We classify communications into three types: *static* communications, *dynamic* communications and *dynamically analyzable* communications. We apply the classification to both point-to-point communications and collective communications. Static communications are communications whose information can be determined at compile time. Dynamically analyzable communications are communications whose information can be determined at runtime without incurring excessive overheads. Dynamic communications are communications whose information can be determined only at runtime. The compiled communication technique is most effective in optimizing static communications. It can be applied to optimize dynamically analyzable communications at runtime with some overheads. Compiled communication is least effective, and usually resorts to traditional communication schemes, in handling dynamic communications. Our study shows that while MPI programs have significantly more dynamic communications in comparison to the programs studied in [4], static and dynamically analyzable communications still account for a large portion of all communications. In particular, the majority of collective communications are static. We conclude that compiled communication can be effective for MPI programs.

The rest of the paper is organized as follows. Section 2 presents the related work. Section 3 summarizes the NAS benchmarks. Section 4 describes the methodology we used in this study. Section 5 presents the results of the study. Section 6 concludes the paper.

II. Related work

The characterization of applications is essential for developing an efficient distributed system and its importance is evidenced by a large amount of existing work [2], [3], [4], [6]. In [2], the overall performance of a large number of parallel architectures was evaluated using the NAS benchmarks. In [3], the NAS benchmarks were used to evaluate two communication libraries over the IBM SP machine. In [6], detailed communication workload resulted from the NAS benchmarks was examined. These evaluations all assume that the underlying communication systems are traditional communication systems and do not classify communications based on whether the communications are static or dynamic. The most closely related work to this research was presented in [4], where the communications in parallel scientific programs were classified as static and dynamic. It was found that a large portion of communications in parallel programs, which include both message passing programs and shared memory programs, are static and only less than 1% of the communications are dynamic. Since then, both parallel applications and parallel architectures evolve, and more importantly, MPI has been standardized, resulting in a large number of MPI based parallel programs. In this work, we focus on MPI programs whose communications have not been characterized in this manner.

III. NAS Parallel Benchmarks

The NAS parallel benchmarks (NPB) [2] were developed at the NASA Ames research center to evaluate the performance of parallel and distributed systems. The benchmarks, which are derived from computational fluid dynamics (CFD), consist of five parallel kernels (EP, MG, CG, FT, and IS) and three simulated applications (LU, BT, and SP). In this work, we study NPB 2.3 [5], the MPI-based implementation written and distributed by NAS. NPB 2.3 is intended to be run with little or no tuning, and approximate the performance a typical user can expect to obtain for a portable parallel program. A detailed description of the benchmarks can be found in [5].

IV. Methodology

In this section, we describe the methodology we used. We will present the assumptions, the methods to classify communications, and the techniques to collect statistical data.

A. Types of communications

The communication information needed by compiled communication depends on the optimizations to be performed. In this study, we assume that the communication information needed is the *communication pattern*, which specifies the source-destination pairs in a communication. Many optimizations can be performed using this information. In a circuit-switched network, for example, compiled communication can use the knowledge of communication patterns to pre-establish connections and eliminate runtime path establishment overheads. When multicast communication is used to realize collective communications, compiled communication can use the knowledge of communication patterns to perform group management statically.

We classify the communications into three types: static communications, dynamic communications, and dynamically analyzable communications. The classification applies to both collective communications and point-to-point communications. Static communications are communications whose pattern information is determined at compile time. Dynamic communications are communications whose patterns can only be determined at runtime. Dynamically analyzable communications are communications whose pattern information can be determined at runtime without incurring excessive overheads. In general, dynamically analyzable communications are usually resulted from communication routines that are invoked repeatedly with one or more symbolic constant parameters. Since the symbolic constants can be determined once at runtime (thus, without incurring excessive overheads) and be used many times in the communications, we distinguish such communications from other dynamic communications. The parameterized communications in [4], that is, communications whose patterns can be represented at compile time using some symbolic constant parameters, belong to the dynamically analyzable communication in our classification.

B. Assumptions about the compiler

The compiler analysis technique greatly affects the compiler's ability to identify static communications. In the study, we emulate the compiler and analyze the programs by hand to mark the communication routines. We make the following assumptions:

- The compiler does not support array analysis. All array elements are treated as unknown variables at compile time.
- The compiler has perfect scalar analysis: it can always determine the value of a scalar if the value can be determined.
- The compiler does not have inter-procedure analysis. Information cannot be propagated across procedure boundaries. The parameters of a procedure are assumed to be unknown. We also assume that simple inlining is performed: procedures that are called only in one place in the program are inlined.

C. Classifying communications

The communication routines in the NAS benchmarks include five collective communication routines: $MPI\_Allreduce$, $MPI\_Alltoall$, $MPI\_Alltoallv$, $MPI\_Barrier$, and MPI_Bcast, and four point-to-point communication routines: MPI_Send, MPI_Isend, MPI_Irecv, and MPI_Recv. Each collective communication routine represents a communication while a point-to-point communication is represented by a pair of MPI_Send/MPI_Isend and MPI_Recv/MPI_Irecv routines. We assume that the benchmarks are correct MPI programs. Thus, MPI_Send/MPI_Isend routines are matched with MPI_Recv/MPI_Irecv routines and the information about point-to-point communications is derived from MPI_Send and MPI_Isend routines.

All MPI communication routines have a parameter called *communicator*, which contains the information about the set of processes involved in the communication. To determine the communication pattern information for each communication, we must determine the processes in the corresponding communicator. Next, we will describe how we deal with the communicator and how we mark each MPI communication routine.

Communicator: The communicators used in the NAS benchmarks either are the MPI built-in communicator, MPI_COMM_WORLD, which specifies all processes for a task, or are derived from MPI_COMM_WORLD using MPI_Comm_split and/or MPI_Comm_dup functions. We assume that MPI_COMM_WORLD is known to the compiler. This is equivalent to assuming that the program is compiled for a particular number of nodes for execution, which is a common practice. A dynamically created communicator is static if we can determine the ranks of all the nodes in the communicator with respect to MPI_COMM_WORLD at compile time. If a communicator is a global variable and is used in multiple communications, it is considered as a dynamically analyzable communicator. The rational to treat such a communicator as a dynamically analyzable communicator is that a communicator typically lasts for a long time in the execution and is usually used in many communications. The overhead to determine the communicator information at runtime is small when amortized over the number of communications that use the communicator. A communicator is considered dynamic if it is neither static nor dynamically analyzable.

For a communication to be static, the corresponding communicator must be static. For a communication to be dynamically analyzable, the communicator can be either static or dynamically analyzable.

MPI_Barrier: The prototype for this routine is *int MPI_Barrier(MPI_Comm comm)*. The communication resulted from this routine is implementation dependent. However, once the compiler determines the communicator $comm$, it also determines the communication pattern for a particular MPI implementation. Thus, the communication is static if $comm$ is static, dynamically analyzable if $comm$ is dynamically analyzable, and dynamic if $comm$ is dynamic.

MPI_Alltoall: The prototype for this routine is *int MPI_Alltoall(void* sendbuf, int sendcount, MPI_Datatype sendtype, void* recvbuf, int recvcount, MPI_Datatype recvtype, MPI_Comm comm)*. This routine results in all nodes in $comm$ sending messages to all other nodes in $comm$. Thus, once $comm$ is determined, the communication pattern for $MPI\_Alltoall$ can be decided. The communication is static if $comm$ is static, dynamically analyzable if $comm$ is dynamically analyzable, and dynamic if $comm$ is dynamic.

MPI_Alltoallv: The prototype for this routine is *int MPI_Alltoallv(void* sendbuf, int *sendcounts, int *sdispls, MPI_Datatype sendtype, void* recvbuf, int *recvcounts, int *rdispls, MPI_Datatype recvtype, MPI_Comm comm)*. The communication pattern for this routine depends on the values of the $sendcount$ array elements. Since we assume that the compiler does not have array analysis in this study, all $MPI\_Alltoallv$ routines are marked as dynamic.

MPI_Allreduce: The prototype for this routine is *int MPI_Allreduce(void* sendbuf, void* recvbuf, int count, MPI_Datatype datatype, MPI_Op op, MPI_Comm comm)*. The communication pattern in this routine is implementation dependent. It is roughly equivalent to a reduction and a broadcast. Once the communicator $comm$ is determined, the communication pattern for a particular implementation can be decided. Thus, the communication is static if $comm$ is static, dynamically analyzable if $comm$ is dynamically analyzable, and dynamic if $comm$ is dynamic.

MPI_Bcast: The prototype for this routine is *int MPI_Bcast(void* buffer, int count, MPI_Datatype datatype, int root, MPI_Comm comm)*. The communication pattern of this routine is $root$ sending a message to all other nodes in the communicator $comm$. Thus, if either $comm$ or $root$ is dynamic, the communication is dynamic. If both $comm$ and $root$ can be determined at compile time, the communication is static. Otherwise, the communication is dynamically analyzable.

MPI_Send: The prototype for this routine is *int MPI_Send(void* buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm comm)*. The analysis of this routine is somewhat tricky: the instantiation of the routine at runtime results in different source-destination pairs for different nodes. For an MPI_Send to be static, all the source-destination pairs resulted from the routine must be determined at compile time. This requires the followings: (1) the communicator, $comm$, should be static; (2) the relation between ranks of the destination and the source nodes should be static; and (3) if there are guard statements (if statement) protecting the routine, the effects of the guard statements should be static. If any of the above is dynamic or dynamically analyzable, the communication is marked as dynamic or dynamically analyzable.

MPI_Isend: The analysis of this routine is similar to that of MPI_Send.

D. Data collection

To collect dynamic measurement of the communications, we instrument MPI operations by implementing an MPI wrapper that allows us to monitor the MPI communication activities at runtime. Notice that we cannot use built-in MPI monitoring utility provided by the existing MPI implementation since we must distinguish among static, dynamically analyzable, and dynamic communications. Such information is not presented in the original MPI routines. To obtain the information, we examine the source code and mark each of the MPI communication routines by hand. In the MPI wrapper, we record all MPI operations with their respective parameters (as well as a field indicating whether the communication is static, dynamic, or dynamically analyzable) in a local trace file. After the execution of the program, we analyze the trace files for all the nodes off-line to obtain the dynamic measurement. Most trace-based analysis systems use a similar approach [7].

V. Results and Implications

Program	communication routines
EP	Static: 4 $MPI\_Allreduce$, 1 $MPI\_Barrier$
CG	Static: 1 SKS $MPI\_Barrier$ Dynamic: 10 MPI_Send
MG	Static: 6 $MPI\_Allreduce$, 9 $MPI\_Barrier$ 6 SKS MPI_Bcast Dynamic: 12 MPI_Send
FT	Static: 2 $MPI\_Barrier$, 2 MPI_Bcast Dynamically analyzable: 3 $MPI\_Alltoall$
IS	Static: 1 $MPI\_Allreduce$, 1 $MPI\_Alltoall$ 1 MPI_Send Dynamic: 1 $MPI\_Alltoallv$
LU	Static: 6 $MPI\_Allreduce$, 1 $MPI\_Barrier$ 9 MPI_Bcast Dynamically Analyzable: 4 MPI_Send Dynamic: 8 MPI_Send
BT	Static: 2 $MPI\_Allreduce$, 2 $MPI\_Barrier$ 3 MPI_Bcast Dynamically Analyzable: 12 MPI_Isend
SP	Static: 2 $MPI\_Allreduce$, 2 $MPI\_Barrier$ 3 MPI_Bcast Dynamically Analyzable: 12 MPI_Isend

TABLE I
STATIC CLASSIFICATION OF MPI COMMUNICATION ROUTINES

Table I shows the static classification of the communication routines in each program. All but one collective communication routine ($MPI\_alltoallv$ in IS) in the benchmarks are either static or dynamically analyzable. In contrast, a relatively larger portion of point-to-point communication routines are dynamic. Figure 1 summarizes the static counts of the different communications. Among the 126 communication routines in all the benchmarks, 50.8% are static, 24.6% are dynamically analyzable, and 24.6% are dynamic. This measurement shows that there are not many MPI communication routines in a program and that using the demand driven method [8], which obtains program information on demand, to analyze the program and obtain information for compiled communication is likely to perform better than using the traditional exhaustive program analysis approach.

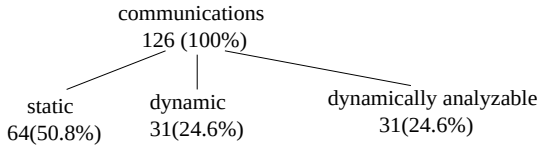

Fig. 1. Summary of static measurement

Prog.	type	num.	num. %	volume	volume %
EP	Static	5	100.0%	12.5KB	100.0%
	D. Ana.	0	0.0%	0	0.0%
	Dynamic	0	0.0%	0	0.0%
CG	Static	1	0.0%	0.03KB	0.0%
	D. Ana.	0	0.0%	0	0.0%
	Dynamic	47104	100.0%	2.24GB	100.0%
MG	Static	100	0.9%	126KB	0.0%
	D. Ana.	0	0.0%	0	0.0%
	Dynamic	11024	99.1%	384MB	100.0%
FT	Static	3	27.3%	0.99KB	0.0%
	D. Ana.	8	72.7%	3.77GB	100.0%
	Dynamic	0	0.0%	0	0.0%
IS	Static	37	77.1%	1.40MB	0.4%
	D. Ana.	0	0.0%	0	0.0%
	Dynamic	11	22.9%	346MB	99.6%
LU	Static	18	0.0%	28.6KB	0.0%
	D. Ana.	12096	1.6%	3.96GB	75.7%
	Dynamic	744036	98.4%	1.27GB	24.3%
BT	Static	7	0.0%	11.1KB	0.0%
	D. Ana.	77280	100.0%	14.1GB	100.0%
	Dynamic	0	0.0%	0	0.0%
SP	Static	7	0.0%	11.1KB	0.0%
	D. Ana.	154080	100.0%	23.7GB	100.0%
	Dynamic	0	0.0%	0	0.0%

TABLE II
DYNAMIC MEASUREMENT FOR LARGE PROBLEMS (SIZE = A) ON 16 NODES

Table II shows the dynamic measurement of the number and the volume for the three types of communications in each of the benchmarks. The results are obtained for large problems (the 'A' class in NPB-2.3 specification) on 16 nodes. The number of communications is the number of times a communication routine is invoked. For collective communications, the invocations of the corresponding routine at different nodes are counted as one communication. For point-to-point communications, each invocation of a routine at each node is counted as one communication. The volume of communications is the total number of bytes sent by the communications. For example, EP has 5 static communications, which transfer 12.5KB data. Different benchmarks exhibit different communication characteristics: in terms of the volume of communications, among the eight benchmarks, CG, MG, and IS are dominated by dynamic communications; EP contains only static communications; FT, BT, and SP are dominated by dynamically analyzable communications. LU has 75.7%

dynamically analyzable communications and 24.3% dynamic communications. In comparison to the results in [4], the NAS parallel benchmarks have significantly less static communications and much more dynamic communications. However, static and dynamically analyzable communications still account for a large portion of the communications, which indicates that compiled communication can be effective if it can be applied to the two types of communications. The results also show that in order for compiled communication to be effective, it must be able to optimize dynamically analyzable communications.

prog.	type	num.	volume	volume %
EP	collective	5	12.5KB	100.0%
	point-to-point	0	0	0.0%
CG	collective	1	0.03B	0.0%
	point-to-point	47104	2.24GB	100.0%
MG	collective	100	126KB	0.0%
	point-to-point	11024	384MB	100.0%
FT	collective	11	3.77GB	100.0%
	point-to-point	0	0	0.0%
IS	collective	33	348MB	100.0%
	point-to-point	15	0.06KB	0.0%
LU	collective	18	28.6KB	0.0%
	point-to-point	756132	5.24GB	100.0%
BT	collective	7	11.1KB	0.0%
	point-to-point	77280	14.1GB	100.0%
SP	collective	7	11.1KB	0.0%
	point-to-point	154080	23.7GB	100.0%

TABLE III

DYNAMIC MEASUREMENT FOR COLLECTIVE AND POINT-TO-POINT COMMUNICATIONS

Since collective communication and point-to-point communication are implemented in a different manner, we will consider collective communication and point-to-point communication separately. Table III shows the number and the volume of collective and point-to-point communications in the benchmarks. The results are for large problems ('A' class) on 16 nodes. The communications in benchmarks EP, FT, and IS are dominated by collective communications, while the communications in CG, MG, LU, BT, and SP are dominated by point-to-point communications.

Table IV shows the classification of collective communications in the benchmarks. The results are obtained for large problems ('A' class) on 16 nodes. As can be seen in the table, the collective communications in six benchmarks, EP, CG, MG, LU, BT, and SP, are all static. Most of the collective communications in FT are dynamically analyzable. Only the collective communications in IS are mostly dynamic. The results show that most of collective communications are either static and dynamically analyzable. The implication is that the compiled communication technique should be applied to optimize the MPI collective communication routines.

Table V shows the classification of point-to-point communications. The results are obtained for large problems ('A' class) on 16 nodes. Since EP and FT do not have point-to-point communication, we exclude these two benchmarks from the table. In comparison to collective communication, more point-to-point communications are dynamic. CG and MG contain only dynamic point-to-point communications. BT and SP contain only dynamically analyzable point-to-point communications. None of the benchmarks has a significant number of static point-to-point communications. However, since a significant portion of point-to-point communications are dynamically analyzable, compiled communication can be effective for point-to-point communications if it is effective for dynamically analyzable communications.

prog.	type	num.	num. %	volume	volume %
EP	Static	5	100.0%	12.5KB	100.0%
	D. ana.	0	0.0%	0	0.0%
	Dynamic	0	0.0%	0	0.0%
CG	Static	1	100.0%	0.03KB	100.0%
	D. ana.	0	0.0%	0	0.0%
	Dynamic	0	0.0%	0	0.0%
MG	Static	100	100.0%	126KB	100.0%
	D. ana.	0	0.0%	0	0.0%
	Dynamic	0	0.0%	0	0.0%
FT	Static	3	27..3%	0.99KB	0.0%
	D. ana.	8	72.8%	3.77GB	100%
	Dynamic	0	0.0%	0	0.0%
IS	Static	22	66.7%	1.37MB	0.4%
	D. ana.	0	0.0%	0	0.0%
	Dynamic	11	33.3%	346MB	99.6%
LU	Static	18	100.0%	28.6KB	100.0%
	D. ana.	0	0.0%	0	0.0%
	Dynamic	0	0.0%	0	0.0%
BT	Static	7	100.0%	11.1KB	100.0%
	D. ana.	0	0.0%	0	0.0%
	Dynamic	0	0.0%	0	0.0%
SP	Static	7	100.0%	11.1KB	100.0%
	D. ana.	0	0.0%	0	0.0%
	Dynamic	0	0.0%	0	0.0%

TABLE IV

DYNAMIC MEASUREMENT OF COLLECTIVE COMMUNICATIONS

prog.	type	num.	num. %	volume	volume %
CG	Static	0	0.0%	0	0.0%
	D. ana.	0	0.0%	0	0.0%
	Dynamic	47104	100.0%	2.24GB	100.0%
MG	Static	0	0.0%	0	0.0%
	D. ana.	0	0.0%	0	0.0%
	Dynamic	11024	100.0%	384MB	100.0%
IS	Static	15	100.0%	0.06KB	100.0%
	D. ana.	0	0.0%	0	0.0%
	Dynamic	0	0.0%	0	0.0%
LU	Static	0	0.0%	0	0.0%
	D. ana.	12096	1.6%	3.96GB	75.7%
	Dynamic	744036	98.4%	1.27GB	24.3%
BT	Static	0	0.0%	0	0.0%
	D. ana	77280	100.0%	14.1GB	100.0%
	Dynamic	0	0.0%	0	0.0%
SP	Static	0	0.0%	0	0.0%
	D. ana.	154080	100.0%	23.7GB	100.0%
	Dynamic	0	0.0%	0	0.0%

TABLE V

DYNAMIC MEASUREMENT OF POINT-TO-POINT COMMUNICATIONS

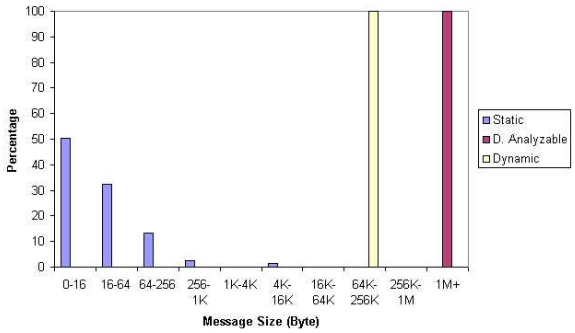

Fig. 2. Message size distribution for collective communications

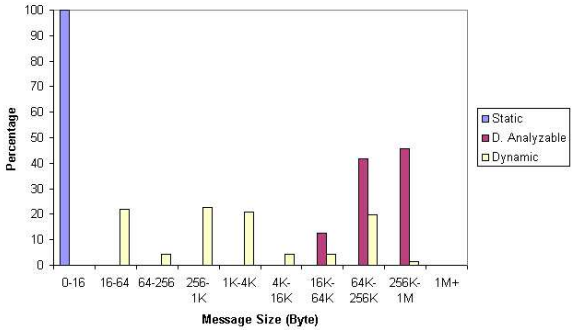

Fig. 3. Message size distribution for point-to-point communications

Message sizes can greatly affect the ways the communications are performed. Figure 2 shows the summary of the message size distribution for collective communications in all the benchmarks. The results are obtained for large problems ('A' class) on 16 nodes. The summary is obtained by first computing the message size distribution in terms of percentage for each range of message sizes in each of the benchmarks. We then give equal weights to all the benchmarks that have the particular communication and calculate the average message size distribution in terms of percentage. The benchmarks only have one dynamically analyzable and one dynamic collective communications, so there is not much distribution for these cases. For static collective communications, the message sizes are mostly small ($< 1KB$). This indicates that static collective communications with small message sizes are important cases for compiled communication.

Figure 3 shows the summary of the message size distribution for point-to-point communications in all the benchmarks. The summary is obtained in a similar manner to that of the collective communications case. The static point-to-point communications have a small message size while dynamic and dynamically analyzable point-to-point communications generally have medium to large message sizes.

We performed experiments with different problem sizes (the 'S' class and the 'W' class in the benchmark) and with different numbers of nodes. Due to the space limitation, we omit the results. The general trend in the message size distribution is similar to the 'A' class on 16 nodes cases except that for a smaller problem size with the same number of processors, the messages tend to be smaller without altering the message size distribution trend and that for a smaller number of processors with the same problem size, the message sizes tend to be larger without altering the trend.

VI. Conclusion

In this paper, we studied the communication characteristics in the MPI implementation of the NAS parallel benchmarks and investigated the effectiveness of compiled communication for MPI programs. The results of this study can also be used by other compiler assisted approaches to improve communication performance. The main conclusions are the followings:

- Static and dynamically analyzable communications account for a significant portion of the communications in the benchmarks, which indicates that compiled communication can be effective if it can optimize these two types of communications.
- The majority of collective communications are static, which indicates that compiled communication should be applied to optimize MPI collective communications. Furthermore, most of static collective communications have small message sizes.
- There is a significant number of dynamically analyzable point-to-point communications in the benchmarks. For compiled communication to be effective in handling MPI point-to-point communications, it must be able to optimize dynamically analyzable communications.

References

[1] The MPI Forum, The MPI-2: Extensions to the Message Passing Interface. *http://www.mpi-forum.org/docs/mpi-20-html/mpi2-report.html*, July, 1997.

[2] D. H. Bailey, T. Harris, R. Van der Wingngaart, W. Saphir, A. Woo, and M. Yarrow, "The NAS Parallel Benchmarks 2.0", *Technical Report NAS-95-010*, NASA Ames Research Center, 1995.

[3] F. Cappello and D. Etiemble, "MPI versus MPI+OpenMP on IBM SP for the NAS Benchmarks," *SC'00: High Performance Networking and Computing Conference*, 2000.

[4] D. Lahaut and C. Germain, "Static Communications in Parallel Scientifi c Programs." *Proceedings of PARLE*, 1994.

[5] *NAS Parallel Benchmarks*, available at http://www.nas.nasa.gov/NAS/NPB.

[6] F. Wong, R. Martin, R. Arpaci-Dusseau, and D. Culler, "Architectural Requirements of Scalability of the NAS Parallel Benchmarks", *SC'99: High Performance Networking and Computing Conference*, 1999.

[7] J.S. Vetter and F. Mueller, "Communication Characteristics of Large-Scale Scientifi c Applications for Contemporary Cluster Architectures", *International Parallel and Distributed Processing Symposium* (IPDPS), April 2002.

[8] Xin Yuan, Rajiv Gupta and Rami Melhem, "Demand-driven Data Flow Analysis for Communication Optimization." *Parallel Processing Letters*, Vol. 7, No. 4, pages 359-370, Dec. 1997.

[9] Xin Yuan, Scott Daniels, Ahmad Faraj and Amit Karwande, "Group Management Schemes for Implementing MPI Collective Communication over IP-Multicast.", The *6th International Conference on Computer Science and Informatics*, Durham, NC, March 8-14, 2002.