An Experimental Assessment of *Express* Parallel Programming Environment

Ishfaq Ahmad*, Min-You Wu, Jaehyung Yang*** and Arif Ghafoor*****

*Hong Kong University of Science and Technology, Hong Kong
*Department of Computer Science, State University of New York, Buffalo, NY
***School of Electrical Engineering, Purdue University, West Lafayette, IN 47907

Abstract

This paper describes a performance evaluation study of Express programming environment on the iPSC/860 hypercube computer. Express allows parallel programs to be developed in a completely portable fashion and is available on most commercially available parallel computers as well as networks of workstations. We have developed a set of benchmarks to make a comprehensive performance assessment of the frequently used communication primitives of Express on hypercube computers. A comparison with the equivalent iPSC/860 primitives is also carried out. In addition, we have developed and benchmarked a suite of applications including three different versions of Gaussian elimination, fast Fourier transform, and the N-body problem. These results help us evaluate the performance of Express and judge its merits against the extra overhead incurred due to portability, and can be useful for designing new generations of portable programming environments. Furthermore, algorithm developers can benefit from these results by using the times required by the basic communication primitives on the iPSC/860 system with and without Express, and can automate performance estimation by using these timings.

1 Introduction

The lack of desired progress in software for parallel computers might be attributed to the fact that there is no unified programming model and that parallelism can be achieved with a variety of paradigms. There also exists the problem of portability - it is distressing to repeat the implementation work on a new machine. A few noteworthy efforts have been made to deal with such issues. These include PVM [14], Linda [1], PICL [6] and Express [12]. These systems allow parallel programs to be developed using C or Fortran by including their message-passing library routines.

Express from Parasoft Corporation is a software programming environment for writing parallel programs for MIMD multiprocessors. Programs can be written using SPMD or pure MIMD paradigms for shared-memory and message passing multiprocessors as well as distributed multicomputers. As illustrated in Figure 1, Express provides a nunber of utilities. These include a communication system for communicating processes, mechanisms for data sharing, reading files, debugging tools, and performance analyzing tools. In addition, it has features such as automatic domain decomposition library which can map physical domain of the problem onto the underlying topology of the parallel or distributed computing system. The performance evaluation tools, using text and graphics, can be effectively used to analyze the run-time performance of the program.

Express has been implemented on a variety of machines including Alliant, BBN Butterfly, nCUBE, Symult, Intel iPSC/2 and iPSC/860 hypercubes, Intel Paragon, Thinking Machine's CM-5, KSR-1, and transputer arrays. The network version of Express allows a network of workstations to be used as a virtual parallel machine. It has been implemented on DEC, HP, IBM/RS6000, SGI,

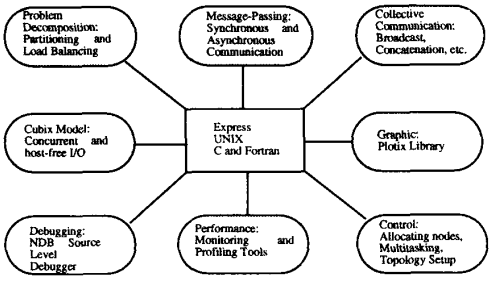

Figure 1: Express utilities

Sun and PC's. Reportedly, over 1000 sites world-wide use Express. From user's stand point, Express is completely portable. That is, a program written for one machine can run on any other machine that has Express without any sort of modification in the source code. The flexibility of Express makes it an attractive tool set for developing and running parallel programs on different platforms. Languages supported by Express include C and Fortran.

A key measure of the usefulness of programming tools like Express is the execution speed of the basic primitives which are frequently used in programs. Since Express provides portability with a rich set of utility functions, the cost is larger overhead incurred due to the translation of high level primitives to the local machine level functions. We have evaluated the performance of some of the basic primitives of Express, running on the iPSC/860. This performance is influenced not only by the implementation of Express primitives but also by the underlying hardware of the two systems. These performance results are important for a number of reasons. In addition, these results can be useful for compiler writers for parallel processors. Algorithm developers can also benefit from these results by understanding the overheads incurred by basic communication primitives on the iPSC/860 hypercube with and without Express, and can automate performance estimation by using these timings [15].

2 History and Overview of Express

The history of Express can be traced back to the Caltech/JPL machines, including the first hypercubes (the Cosmic Cube and Mark II), developed in the early eighties [13]. Initially, a so called "operating system" known as CrOS (Crystalline Operating system) consisting of a few message passing primitives was the only support that was available to derive those machines [4]. The need for efficient solutions of a number of scientific problems led to the incorporation of a number of message passing primitives, such as broadcast and concatenation, into CrOS. The advanced version of this operating system, called CrOS III, was implemented on the Mark III hypercube [5]. To remove from the user the burden of writing host programs, a system known as "c3po" was developed which provided a shell based generic host program. A more so-

0-8186-6902-0/95 $4.00

phisticated version of the system, called Cubix, allowed opening, reading and writing files and interfacing with the user in a host-free programming style. By the end of 1987, the group of researchers who developed these systems started Parasoft Corporation. Express, one of the commercial product of Parasoft Corporation is an integrated software package developed as a result of their research efforts at Caltech.

3 Test Environment

For performance evaluation, we used two commercially available parallel computers from Intel, the iPSC/860 at Caltech's Concurrent Supercomputing Center. The operating system on nodes is NX (Node eXecutive) developed by Intel [7]. Details of the iPSC/860 node architectures can be found in [9]. The summary of some of the architectural features of the computer we used is given in Table 1.

Some performance results showing the computational operations on the iPSC/860 can be found in [9]. All of our programs have been written in Fortran. For communications tests, every point in each test is the result of taking the average of a large number of repetitions. For small data sizes, we have used 1000 repetitions to improve the accuracy, while for larger messages, the number of repetitions has been varied from 100 to 200. The node clock has a resolution of 1 microsecond.

Table 1 : Configurations of the iPSC/860 used in the experiments.

Number of nodes	32
Node CPU	i860
Clock Frequency	40 MHz
Cache Memory/node	8 Kb
Main Memory/node	8 Mb
Express Version	3.1
iPSC Node O/S	NX/2 (rel. 3.2)
Express Version	3.1

4 Communication Performance

The basic communication tests include one-to-one communication among nearest neighbors, multi-hop communication, and broadcast. The results of our benchmark programs for these tests are provided in the following sections.

4.1 One-to-One Communication

For measuring the communication speed between two nodes, we performed the standard echo test. In the echo test, the communication time between sending and receiving processors is measured by starting a clock at the sending processor and then invoking the send and receive routines to send out a message and wait for a reply. On the destination processor, receive and send routines are used to echo this message back to the sending processor. This process is repeated n number of times and the clock is then stopped. The communication time is then taken as elapsed time divided by $2n$.

It is well known that on the iPSC/860 hypercube, there are two different protocols used for sending small and large messages [2], [3]. The time for one-to-one communication can be described by the following equation.

$$T_{startup} + B \times T_{transmission} + h \times d$$

where B is the message length in number of bytes, T_{st} is the latency incurred due to the time required to set up the communication request, $T_{transmission}$ is the transmission time for one byte, d is the hop distance between the sending and receiving nodes and h is the extra time for each traversing each hop. These parameters for the iPSC/860, with Express and NX, are shown in Table 2 for $0 < B \leq 100$ and $B > 100$, respectively.

From Table 2, it can be seen that the start-up latency, $T_{startup}$, for sending small messages on the iPSC/860 is 62.5 microseconds using NX and 80.05 microseconds using Express- a difference of

Table 2 : Communication parameters for node-to-node message passing.

	Small messages		Large messages	
	Express	NX	Express	NX
$T_{startup}$	80.05	62.5	160.5	147.0
$T_{transmission}$	0.431	0.421	0.394	0.394
h	10.12	10.0	30.38	29.29

18 microseconds. We also observe that the difference in $T_{transmission}$ of Express and NX is not very large. Using Express, additional 0.12 microseconds account for each hop distance between two nodes. For messages of more than 100 bytes, the difference in $T_{startup}$ using NX and Express is 13.5 microseconds. $T_{transmission}$ on the iPSC/860 is the same for NX and Express. The difference in h for Express and NX on both systems is negligible.

Figure 1(a) shows the plots of times required for one-to-one communication between directly connected processors for message sizes ranging from 0 to 200 bytes. Inspection of these plots reveals that Express is slower than NX by a factor of about 14 to 20%. Figure 1 (b) and Figure 1 (c) show the result for message sizes of 200 to 1000 bytes and 1000 to 16000 bytes, respectively. Relatively, the overhead of using Express is negligible and the spacing between the plots of NX is constant. Figure 1 (d) illustrates plots of times required for communication between nodes that are at 2, 3, 4 and 5 hop distance. Using NX, the spacing of curves is 10 microseconds for messages of length up to 100 bytes and about 30 microseconds for messages of length greater than 100. As described earlier, the amount of spacing is slightly larger for Express (1 to 2%).

4.2 The Broadcast Operation

Broadcast is performed to do broadcasting operations among the nodes and to and from the host node. It is used in various linear algebra algorithms, matrix-vector multiplication, matrix-matrix multiplication, LU-factorization, [8] etc. Figure 3(a) to Figure 3(d) show the timings of broadcasting for various message sizes. The number of nodes in this case has been chosen as 4. The inspection of these figures indicates that, for small messages, Express is about 10% slower than NX. For medium messages, compared to NX, the broadcast operation of Express is again about 10% slower. For very large messages, the difference is less than 1%. From Figure 3(d), we observe that the curves for Express are shifted a little bit upwards compared to NX but are equally spaced implying that extra overhead of Express's broadcast primitive is not a function of number of processors.

4.3 Global Communication

In global communication, all the nodes participate in some operation. The reduction operations and concatenation operations are two examples that require global communication. Global opera-

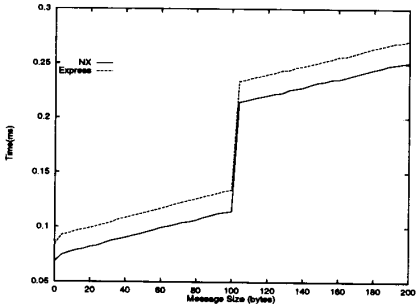

Figure 2(a): Times for one-to-one communication for small size messages with Express and NX primitives..

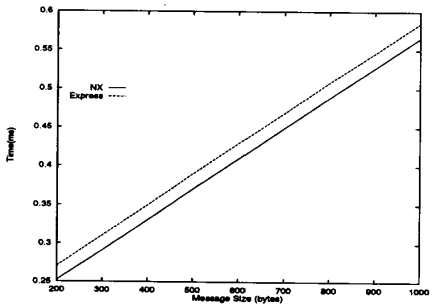

Figure 1(b): Times for one-to-one communication for medium size messages with Express and NX primitives.

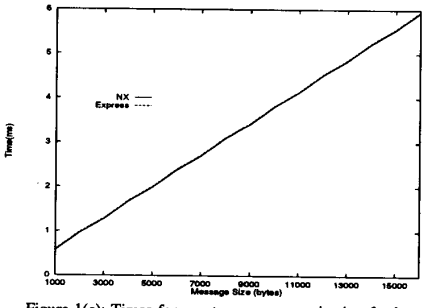

Figure 1(c): Times for one-to-one communication for large size messages with Express and NX primitives.

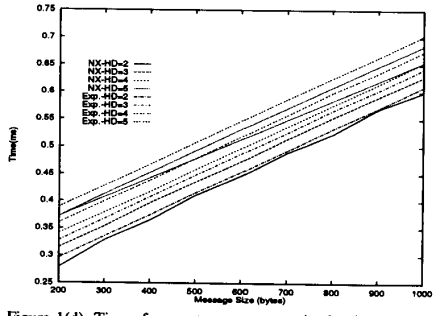

Figure 1(d): Times for one-to-one communication between nodes at various hop distances (HD) with Express and NX primitives.

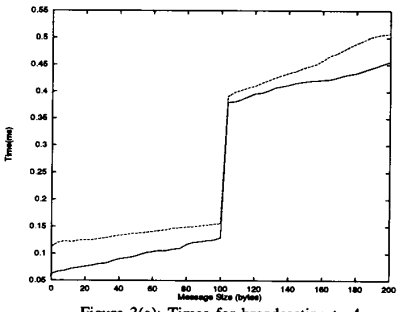

Figure 3(a): Times for broadcasting to 4 processors for small size messages.

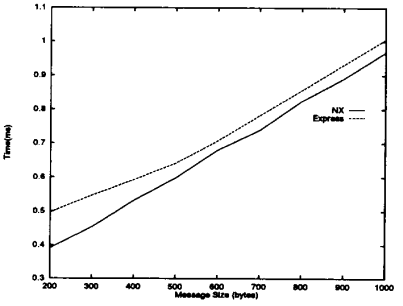

Figure 2(b): Times for broadcasting to 4 processors for medium size messages.

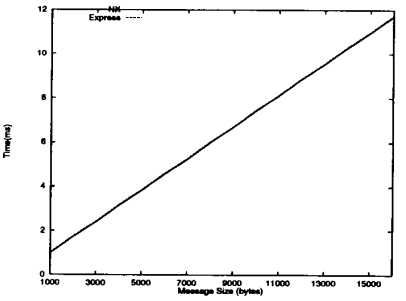

Figure 2(c): Times for broadcasting from one processor to 4 processors for large size messages

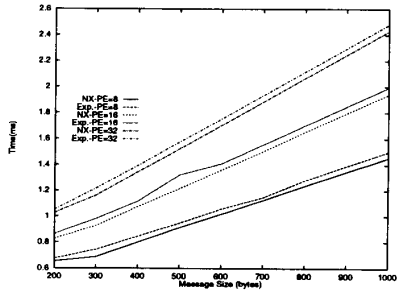

Figure 2(d): Times for broadcasting from one processor to 8, 16 and 32 processors for medium size messages

tions for producing reduction across all the nodes are frequently used in parallel programs. A reduction operation takes as input a value in each processor and outputs a single value in every processor. There can be many types of reduction operations such as *add, prod, max, min, and, or, xor*, etc. While NX provides different calls for each of such operations, Express has a unified function for performing reduction operations. The timing results for a selected set of data points for global sum are provided in Table 3. The data set includes times to perform these operations on 1, 2, 4, and 26 words. The results are repeated for 8, 16 and 32 processors.

Table 3 Times for global sum (milliseconds).

		Global Sum		Global Max.		Global Product	
Processors	Words	Express	NX	Express	NX	Express	NX
4	1	0.404	0.265	0.409	0.291	0.409	0.274
	2	0.410	0.273	0.415	0.293	0.414	0.278
	4	0.424	0.277	0.428	0.296	0.426	0.278
	26	1.098	0.562	1.101	0.586	1.197	0.562
8	1	0.604	0.422	0.606	0.445	0.607	0.444
	2	0.613	0.427	0.616	0.448	0.617	0.446
	4	0.636	0.431	0.638	0.465	0.638	0.458
	26	1.819	1.045	1.829	1.107	1.823	1.101
16	1	0.797	0.597	0.809	0.606	0.811	0.598
	2	0.811	0.604	0.822	0.619	0.824	0.606
	4	0.843	0.609	0.854	0.633	0.854	0.610
	26	2.447	1.722	2.462	1.770	2.457	1.722
32	1	1.033	0.766	1.011	0.783	1.083	1.138
	2	1.053	0.774	1.031	0.789	1.159	1.289
	4	1.091	0.788	1.073	0.801	1.305	1.326
	26	3.138	2.305	3.113	2.375	4.677	4.084

5 Evaluation with an Application Benchmark Suite

In order to compare the performance of Express and NX systems with real applications, we have implemented an application benchmark suite. The applications in the benchmark suite include three different versions of Gaussian elimination, fast Fourier transform and the N-body problem.These algorithms have been coded using Express and NX primitives on the iPSC/860. We have obtained the execution times of these programs on 1, 2, 4, 8, 16 and 32 nodes. The execution time of a program is taken as the maximum of all the nodes. The results are given in the following sections.

5.1 Gaussian Elimination (Row-block Partitioning)

The three versions of Gaussian elimination for solving linear equations are based on partial pivoting algorithm. We have used different techniques to partition the data across processors. As a result, the algorithms used in these versions are quite different. In this section, we present the results of the first version of Gaussian elimination. In this version, the data has been partitioned in blocks of rows, that is, an equal number of contiguous rows of the coefficient matrix are assigned to each processor.

The execution times of Gaussian elimination with row-block partitioning using Express with various matrix sizes and number of processors are shown in Table 5. The corresponding execution times using NX are shown in the parenthesis. This table also indicates the times for serial execution of Gaussian elimination using one processor. As can be seen, the execution times for the Express version exhibit gain in speedup if the number of processors are increased from 1 to 16. The speedup is better if the matrix size is large. However, the execution times start increasing if 32 processors are used. On the other hand, the NX version still yields some speedup with 32 processors. Express performs poorly with large number of processors. One can also observe that Express performs better for larger data sizes. The poor performance of Express with large number of processors is due to a number of factors. First, the algorithm used in this implementation makes an extensive use of broadcast operations for sending the pivot row to other processors. Second, no optimizations are made in communication calls to exploit the hypercube topology. As a result, the algorithm uses a number of global operations which are quite slow with Express. Finally, for small matrices and large number of processors, the grain sizes become small. Then the impact of large overhead in Express makes it significantly slower than NX.

Table 4 Timings (seconds) for row-partitioned Gaussian elimination. The values in parenthesis are the execution times using NX.

Matrix \ PEs	2	4	8	16	32
256 x 256	3.75 (3.64)	2.31 (2.22)	1.81 (1.47)	2.11 (1.06)	3.46 (0.93)
3844 x 384	12.99 (12.77)	6.79 (6.64)	4.72 (4.02)	4.44 (2.59)	6.87 (2.15)
512 x 512	31.13 (30.08)	15.65 (15.49)	9.20 (8.55)	8.50 (5.55)	11.38 (4.10)
640 x 640	62.62 (61.14)	31.06 (30.97)	17.49 (16.20)	13.96 (9.80)	17.37 (6.89)
768 x 768	108.24 (106.5)	54.71 (54.2)	28.66 (27.91)	20.77 (16.74)	24.69 (10.82)
896 x 896	182.77 (181.9)	89.06 (88.5)	46.53 (44.8)	31.19 (24.60)	34.01 (15.77)
1024 x 1024	261.94 (260.4)	131.08 (130.4)	66.21 (63.9)	43.23 (34.96)	45.56 (21.99)

5.2 Gaussian Elimination (Column-block Partitioning)

In this version of Gaussian elimination, the data is partitioned across processors in terms of blocks of columns. Table 5 shows the executions times for this algorithm. It can be noticed that the execution times in this case are significantly greater than those of the row-block partitioning algorithms. This is due to the fact that in the row-block partitioning algorithm, the determination of the pivoting row is done in parallel. On the other hand, column-block partitioning algorithm performs this step serially. Again, Express is shown to perform well for large matrices if the number of processors is between 2 and 16, but performs poorly for 32 processors. For smaller matrices, Express is 3 to 4 times slower than NX.

Table 5 Timings (seconds) for column-block partitioned Gaussian elimination. The values in parenthesis are the execution times using NX.

Matrix \ PEs	2	4	8	16	32
256 x 256	7.03 (6.26)	3.64 (3.18)	2.78 (1.84)	3.79 (1.20)	5.35 (1.13)
3844 x 384	23.92 (23.07)	11.57 (6.78)	6.75 (5.03)	7.64 (3.17)	12.30 (2.61)
512 x 512	47.55 (45.97)	27.48 (23.02)	14.26 (11.89)	13.19 (6.61)	20.25 (5.12)
640 x 640	110.18 (102.6)	56.49 (51.54)	28.62 (25.79)	20.75 (12.14)	30.30 (8.28)
768 x 768	182.78 (178.6)	98.72 (90.62)	50.24 (46.86)	30.96 (24.17)	42.69 (12.96)
896 x 896	290.50 (287.4)	151.39 (147.9)	79.62 (74.66)	49.90 (37.77)	57.52 (19.22)
1024 x 1024	408.08 (404.5)	226.55 (222.8)	118.61 (110.4)	65.52 (51.09)	80.61 (31.20)

5.3 Gaussian Elimination (Column-scatter Partitioning)

In this version, the data is partitioned using cyclic distribution of the columns of the coefficient matrix. The results are given in Table 5 which indicate that this algorithm performs better than column-block partitioning but worse than row-block partitioning. This is due to the fact that column-scatter partitioning can balance load well, and as a result improve performance. However, it requires excessive exchanges of messages which can delay the execution. When the matrix size is small and the number of processors is large, the benefit of load balancing is small compared to extra cost of communication.

Table 6 Timings (seconds) for column-scattered partitioned Gaussian elimination. The values in parenthesis are the execution times using NX.

Matrix \ PEs	2	4	8	16	32
256 x 256	4.39 (4.07)	2.59 (2.21)	2.66 (1.44)	3.76 (1.25)	6.34 (1.22)
3844 x 384	14.50 (13.92)	7.68 (6.68)	5.93 (3.79)	7.52 (3.00)	12.26 (2.76)
512 x 512	33.90 (31.98)	17.73 (16.23)	11.23 (8.90)	12.87 (5.81)	20.10 (5.04)
640 x 640	65.61 (63.27)	34.13 (32.10)	19.80 (16.14)	19.96 (9.83)	30.01 (8.14)
768 x 768	112.71 (110.67)	58.34 (55.59)	34.85 (28.61)	29.16 (15.42)	42.14 (12.46)
896 x 896	176.13 (172.06)	92.06 (88.42)	52.07 (45.01)	41.02 (22.98)	56.66 (17.46)
1024 x 1024	264.72 (260.42)	137.02 (131.85)	75.46 (63.98)	57.84 (33.26)	79.37 (23.91)

5.4 Fast Fourier Transform

For the fast Fourier transform (FFT), we used the algorithm given in [5] with modification. For N points, the algorithms's complexity is $N(\log N)$. The two dimensional matrix is partitioned across nodes in a row orientation style. The algorithm works by transforming the rows first, then transposing the rows with columns, transforming the columns and so on. We applied vector communication and reduced repeated computation.

Table 7 shows the execution times for inputs ranging in size from 2k to 64k using various number of processors. The performance differences between Express and NX are found to be very small. This is due to the fact that FFT has a very regular communication pattern. Due to smaller number of communication operations, the problem granularity is large and the effect of extra overhead of Express is substantially low.

Table 7 Timings (seconds) for fast Fourier transform.

Matrix \ PEs	2	4	8	16	32
64k	18.989 (18.507)	12.271 (12.036)	8.768 (8.688)	6.956 (6.939)	6.078 (5.987)
32k	8.822 (8.614)	5.738 (5.658)	4.115 (4.107)	3.325 (3.262)	2.904 (2.818)
16k	4.079 (3.988)	2.667 (2.647)	1.932 (1.915)	1.576 (1.522)	1.367 (1.314)
8k	1.887 (1.864)	1.243 (1.238)	0.911 (0.893)	0.739 (0.711)	0.645 (0.615)
4k	0.862 (0.857)	0.577 (0.568)	0.424 (0.411)	0.338 (0.328)	0.299 (0.285)
2k	0.395 (0.394)	0.265 (0.262)	0.197 (0.190)	0.160 (0.152)	0.140 (0.13)

5.5 The N-body Problem

The program for the N-body problem has been written using the algorithm reported in [5]. The algorithm used in this program is the simple $O(N^2)$ algorithm and not the more optimized $O(N(\log N))$ approach. The execution times for this problem are shown in Table 8 using various number of bodies and processors. The difference in the performance of Express and NX is insignificant if small number of processors are used. For 32 processors, the performance difference is large for small problem sizes but decreases as the problem size is increased.

Table 8 Timings (seconds) for the N-body problem

Matrix \ PEs	2	4	8	16	32
16k	1401.2 (1342.9)	694.64 (674.4)	350.30 (339.8)	177.67 (170.0)	89.88 (85.34)
8k	342.19 (331.5)	171.23 (167.3)	86.99 (84.11)	44.47 (42.34)	24.01 (22.42)
4k	86.50 (84.16)	44.00 (42.51)	22.27 (21.58)	12.29 (11.43)	8.19 (6.32)
2k	22.99 (22.25)	11.36 (11.20)	5.90 (5.42)	3.35 (2.89)	2.23 (1.61)
1k	6.20 (5.69)	2.97 (2.97)	1.69 (1.49)	0.95 (0.81)	0.67 (0.53)
512	1.52 (1.49)	0.87 (0.85)	0.48 (0.41)	0.31 (0.26)	0.32 (0.20)

6 Conclusions

In this paper, we have examined one such popular software, known as Express, by benchmarking its basic primitives on the iPSC/860. For one-to-one communication, the difference between the timings of Express and NX on the iPSC/860 is negligible and is only about 15 to 20 microseconds. The difference in the transmission times of Express and NX is not very large and the main difference lies in latencies. For most of the primitives, the relative performance of Express, compared to NX, decreases if data size is increased. In other words, the relative degradation in performance becomes negligible for very large data sizes. However, if an algorithm requires a large number of small messages, its execution time with Express can be considerably higher as compared to NX. Express also tends to become slower with an increase in the number of processors. In general, when programming with small granularity, Express shows significantly worse performance, but with large granularity, Express and NX will perform almost the same. From the performance results on the benchmark suite, we also notice that Express does not scale very well with the increase in number of processors. However, considering its functionality and advantages of portability, the overhead of Express appears not too large and is affordable.

References

[1] S. Ahuja, N. Carriero and D. Gelernter, "Linda and Friends," *IEEE Computer*, no. 8, August 1986,

[2] S. Bokhari, "Communication Overhead on the Intel iPSC/860 Hypercube," ICASE Interim Report 10 182055, NASA Langley Research Center, Hampton, VA, May 1990.

[3] T. H. Dunigan, "Performance of the Intel iPSC/860 and Ncube 6400 Hypercubes," *Parallel Computing*, No. 17, 1991, North Holland, pp. 1285-1302.

[4] J. Flower and A. Kolawa, "A Packet History of Message Passing Systems," *Parasoft Corporation*, 1992.

[5] G.C. Fox, M.A. Johnson, G.A. Lyzenga, S.W. Otto, J.A. Salmon, and D.A. Walker, *Solving Problems on Concurrent Processors*, Volum I, Prentice Hall, 1988.

[6] G.A. Geist, M.T. Heath, B.W. Peyton and P.H. Worley, *A User's Guide to Picl, a Portable Instrumented Communication Library*, Technical Report ORNL/TM-11616, Oak Ridge National Laboratory, Oak Ridge, TN, Oct 1991.

[7] Intel Corporation, *iPSC/2 and iPSC/860 Programmers Reference Manual*, June 1990.

[8] S. L. Johnson and C.T. Ho, "Optimal Broadcasting and Personalized Communication in hypercubes," *IEEE Tran. on Computers*, C-38, No. 9, Sept. 1989, pp. 1249-1268.

[9] S.A. Moyer, "Performance of the iPSC/860 Node Architecture," Tech. Report IPC-TR-91-007, Institute of Parallel Computations, University of Virginia, May 1991.

[10] S.F. Nugent, "The iPSC/2 Direct-Connect Communications Technology," *The Third Conference on Hypercube Concurrent Computers and Applications*, Vol. I, 1988, pp. 51-60.

[11] Parasoft Corporation, *Express, Fortran User's Guide*, 1990.

[12] Parasoft Corporation, *Express Introductory Guide* version 3.2, 1992.

[13] C.L. Seitz, "The Cosmic Cube." Communications of the ACM, No. 28, 1985, pp. 22-33.

[14] V. Sunderam, "PVM: A Framework for Parallel Distributed Computing," Concurrency: Practice & Experience, Vol. 3, No. 4, Dec 1990, pp. 315-339.

[15] J. Yang, I. Ahmad and A. Ghafoor, "Estimation of Execution Times on Heterogeneous Supercomputer Architectures," *Int'l Conference on Parallel Procesing*, 1993, pp. I-219-226.