Express versus iPSC/2 Primitives: A Performance Comparison

Ishfaq Ahmad
Min-You Wu

CRPC-TR91147
April 1991

Center for Research on Parallel Comput
Rice University
P.O. Box 1892
Houston, TX 77251-1892

SCCS-77

CRPC-TR91-147

Express versus iPSC/2 Primitives:

A Performance Comparison

(Draft)

Ishfaq Ahmad and Min-You Wu

April 1991

Syracuse Center for Computational Science

Syracuse University

111 College Place

Syracuse, New York 13244-4100

< sccs@npac.syr.edu >

(315) 443-1723

Table of Contents

1. Introduction

EXPRESS, developed by Parasoft [1], is a software programming environment for writing parallel programs for homogeneous MIMD multiprocessors. It provides a communication system for communicating processes, mechanisms for data sharing, reading files, debugging tools, and performance analyzing tools. An important feature of EXPRESS is that these functionalities are carried out in a user transparent manner. The flexibility of EXPRESS makes it an attractive tool set for developing parallel programs. In addition, it has nice features which allow an application to use appropriate utilities. One of such features is automatic domain decomposition library which can map physical domain of the problem to the underlying topology of the parallel computing system. The performance evaluation tools, using text and graphics, can be effectively used to analyze the run time performance of the program.

The most important feature of EXPRESS is that it is portable. This leads to two advantages. The first advantage is that it can be implemented on a variety of machines such as NCUBE–1, NCUBE–2, Symult, and Intel iPSC/2 and iPSC/860 hypercubes, transputer arrays and shared memory BBN Butterfly system. In addition, it can be implemented on various types of workstation networks. The second advantage of portability is that programs written under EXPRESS for one machine can be run on another machine and the user does not have to worry about the hardware. The languages supported by EXPRESS are C and FORTRAN.

The communication primitives for message passing under EXPRESS make use of asynchronous communication which include exwrite and exread functions at the source and destination nodes, respectively. The EXPRESS kernel provides intermediate buffering and routing. A subset of Express environment is CrOS III which provides synchronous communication system [4]. The third subset of EXPRESS is CUBIX system which allows nodes to perform concurrent I/O.

As depicted in Figure 1, the EXPRESS environment consists of three layers. The lowest level consists of utilities for controlling the hardware. The medium level provides support for problem partitioning and communication between the nodes, and between the node and control processor. The highest level contains the facilities for node programs to perform I/0 to the host operating system.

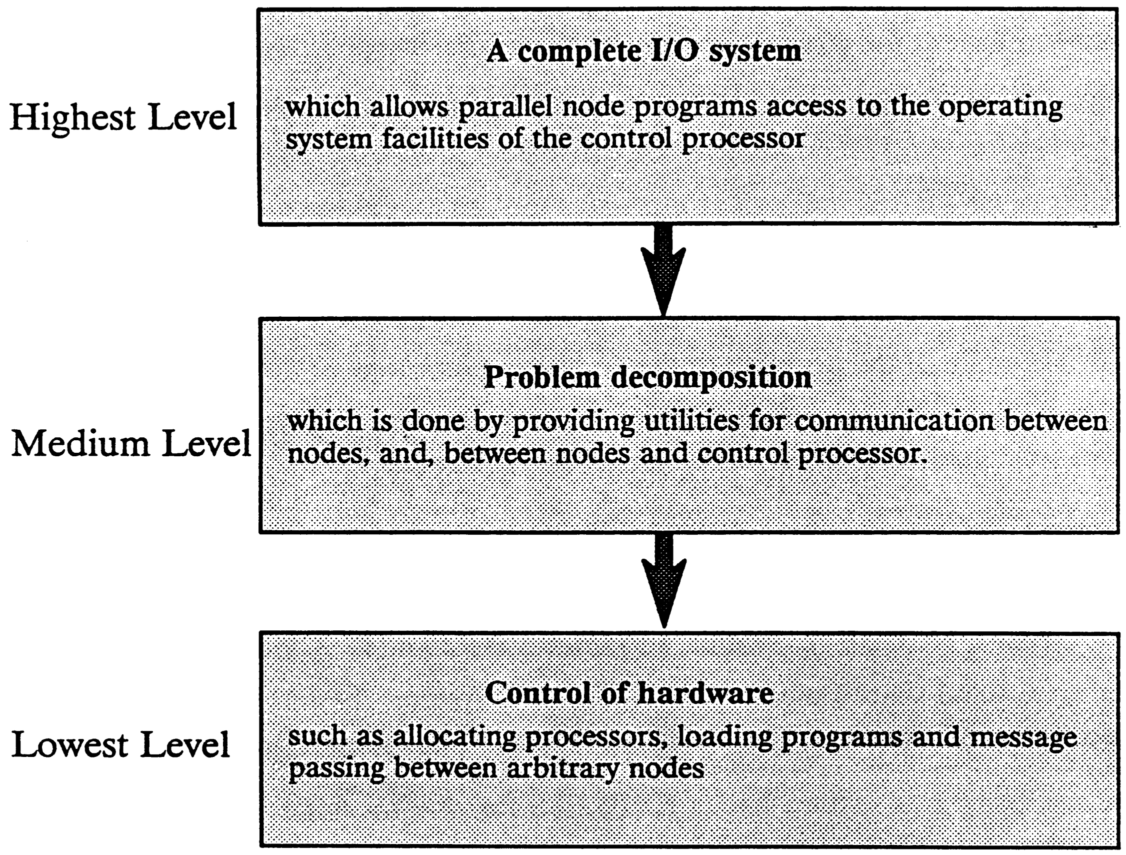
Highest Level
A complete I/O system
which allows parallel node programs access to the operating system facilities of the control processor
Medium Level
Problem decomposition
which is done by providing utilities for communication between nodes, and, between nodes and control processor.
Lowest Level
Control of hardware
such as allocating processors, loading programs and message passing between arbitrary nodes

Figure 1: Three layers of EXPRESS

2. The *EXPRESS* Programming Models

EXPRESS allows parallel programs to develop in the following four different styles.

(1) The conventional ***Host–Node*** Style in which the control part of the program appears in the host program and the computation intensive part appears in node programs.

(2) The *Cubix* environment in which only node program is written and it interfaces to the outside world through graphics and text server.

(3) The *Profile* environment which can be added to both Host–Node and Cubix programming models.

(4) The *Plotix* environment which provides extra graphics features to analyze program profiling.

3. The EXPRESS Utilities

EXPRESS provides a number of utilities which can be classified as follows.

(1) User Commands:

- Profiling Tools
  - I. ctool (Analyze communication profile data).
  - II. xtool (Analyze execution profile data).
  - III. etool (Analyze events profile data).
- System administration such as configuration, reboot and reload.
- User's usage of system resources.
- Source level debugger (communication, execution and events).

(2) Compilers:

C and Fortran Compilers for hypercubes.

C and Fortran Compilers for transputers.

(3) Initialization:

- Starting up *EXPRESS* system.

(4) Processor Control:

- Allocation and de–allocation of processors.
- Loading program into all nodes.
- Loading program into selected nodes.

(5) Interprocessor communication:

- Blocking communication among nodes e.g read and write a message, read and write a vector.
- Global communication.

e.g broadcast, synchronous data exchange, global concatenation, global reduction operations, synchronization, etc.

- Asynchronous communication.

  e.g non–blocking read and write.

- Hardware dependent communication.

  e.g reading and writing data on a channel.

(6) Decomposition Tool:

- Initializing decomposition system.
- Map user domain coordinates to processor number.
- Distributing data to processors.
- Distribute processors on user domain.
- Distributing data to processors.
- Determining run–time environment.

(7) I/O Routines

(8) Multi–tasking

(9) Multi–Hosting

(10) Graphics

(11) Utilities Routines

4. The Performance of Some EXPRESS and Equivalent iPSC/2 Routines

An important issue associated with programming tools is the execution speed of the basic primitives which are frequently used in programs. Examples of such primitives include communication between nodes, communication between host and nodes, global reduction operations, data broadcast, concatenation etc. We have evaluated the performance of some of the basic primitives of EXPRESS, running on iPSC/2. To make comparison with iPSC/2, we also benchmarked equivalent iPSC/2 primitives. In those cases where equivalent primitives did not exist, we wrote routines that can perform the same function.

Our benchmark model consists of two host programs – one for EXPRESS and one for iPSC/2, and one node program for each primitive. Separate node programs are written for EXPRESS and iPSC/2. A host program consists of a menu through which a selection can be made to test one particular primitive. After the selection has been made, additional information such as the number of nodes, source and destination for communication, the choice of host–node communication or node–node communication, the message size, must be provided. The host program selects the appropriate node program and loads it into the nodes. It passes the additional information to the node program and sets the appropriate environment. Each tested primitive is timed with a loop which repeats the same test up to 200 times. At the end of each loop step, all timing information is recorded and appropriate parameters are reset where necessary. Synchronization is done where it is deemed necessary. At the end of the loop, the timing of each test step is sent back to the host which performs statistical analysis on it by calculating the mean value and variance. If the variance is very high, the same test is repeated with a higher number of repetitions. The timing results, including the mean and variance, for the host and node operations are stored in two different files.

Since the message passing protocols on iPSC/2 are different for large and small messages, our tests included both small and large messages. The size of large messages was varied from 128 to 8192 words and the size of small messages was kept fixed as 10 words. The data type of message objects is real type for all tests, which consists of 4 bytes. The actual message size, therefore, is message size multiplied by 4. All timing values are given in milli–seconds. To compare EXPRESS with iPSC/2 primitives, a timing ratio of EXPRESS primitive versus iPSC/2 primitive is calculated and provided in the corresponding table.

4.1 Send/Recv. between Host and Node

The communication between host and node is an important routine that is mostly used for sending initial data and other information from host to nodes and getting the results from nodes to host. It is also used for I/O between nodes and the file system, through the host. The communication can be blocking as well as non–blocking. We considered only the blocking type. The same routines can be used for communication between nodes and between nodes and host. Following are the EXPRESS and iPSC/2 routines for blocking communication.

EXPRESS routines :

INTEGER FUNCTION KXREAD(BUF, LENGTH, SRC, TYPE)
INTEGER FUNCTION KXWRITE(BUF, LENGTH, DEST, TYPE)

Equivalent iPSC/2 routines:

SUBROUTINE CRECV(TYPE, BUF, LENGTH)
SUBROUTINE CSEND(TYPE, BUF, LENGTH, DEST, PID)

Timing results :

Table 1 contains the execution time of EXPRESS KXREAD and KXWRITE and iPSC/2 CSEND and CRECV for host to node communication, for various message sizes. We measured timing for both way communication, that is, sending data from host to node and measuring node time and receiving data from node to host and measuring node's sending time.

Table 1: Node Time to receive and send data to host (ms)

	EXPRESS		iPSC/2		Timing Ratio	
Message Size	recv.	send	recv.	send	recv.	send
10	1.93	0.36	1.46	0.34	1.32	1.06
128	3.16	9.98	2.91	5.91	1.04	1.69
256	3.30	10.91	3.29	5.97	1.00	1.83
512	3.86	11.26	3.54	6.31	1.09	1.78
1024	4.93	11.93	4.18	7.24	1.18	1.65
2048	6.82	13.51	5.83	8.84	1.17	1.53
4096	11.16	16.97	9.08	12.01	1.23	1.41
8192	19.34	23.33	15.60	18.48	1.24	1.26

The results indicate that for both EXPRESS and iPSC/2, the time to send data from node to host is greater than the time to receive data from host to node. We also observe that iPSC/2 performs better by a maximum factor of 1.322 for receiving the data and by a factor of 1.827 for sending data.

4.2 Send/Recv. between Nodes

We used the same routines for node to node communication that were used for host to node communication.

<u>Timing Results :</u>

Again, we measured two way timing, that is, the sending time at the sending node and receiving time at the receiving node. Table 2 summarizes these results along with timing ratios for both receive and send. To present these results pictorially, plots of these values for receive and send are given in Figure 2 and Figure 3.

Table 2.1: Timing results (ms) for the node to node communication
(One to one communication)

	EXPRESS		iPSC/2		Timing Ratio	
Message Size	recv.	send	recv.	send	recv.	send
10	0.39	1.05	0.35	0.40	1.11	2.63
128	0.77	1.81	0.77	0.95	1.00	1.91
256	0.95	2.10	0.94	1.14	1.06	1.84
512	1.30	2.69	1.25	1.47	1.04	1.83
1024	2.01	3.89	1.96	2.30	1.03	1.69
2048	3.53	6.25	3.44	3.75	1.03	1.67
4096	6.50	10.92	6.33	6.76	1.03	1.62
8192	12.38	21.94	12.25	12.50	1.01	1.76

From Table 2.1 , and Figure 2 and Figure 3, we notice that the performance of both systems is almost the same for receiving data. However, iPSC/2 outperforms EXPRESS for sending data.

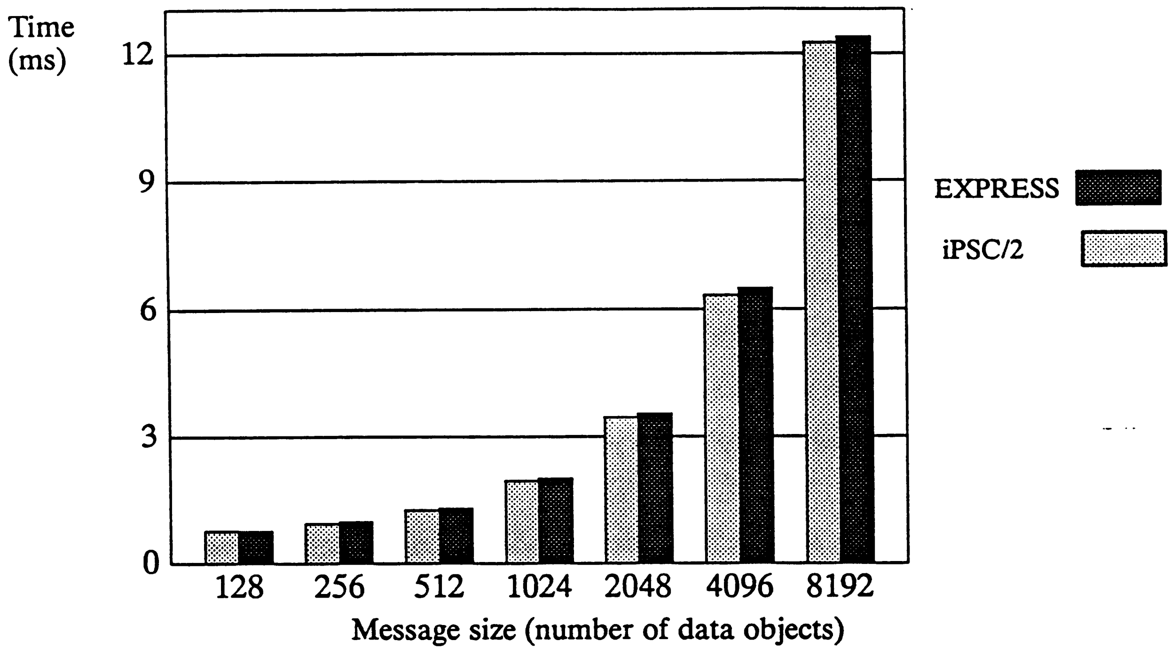

Figure 2: EXPRESS and iPSC/2 time for receive operation versus message size

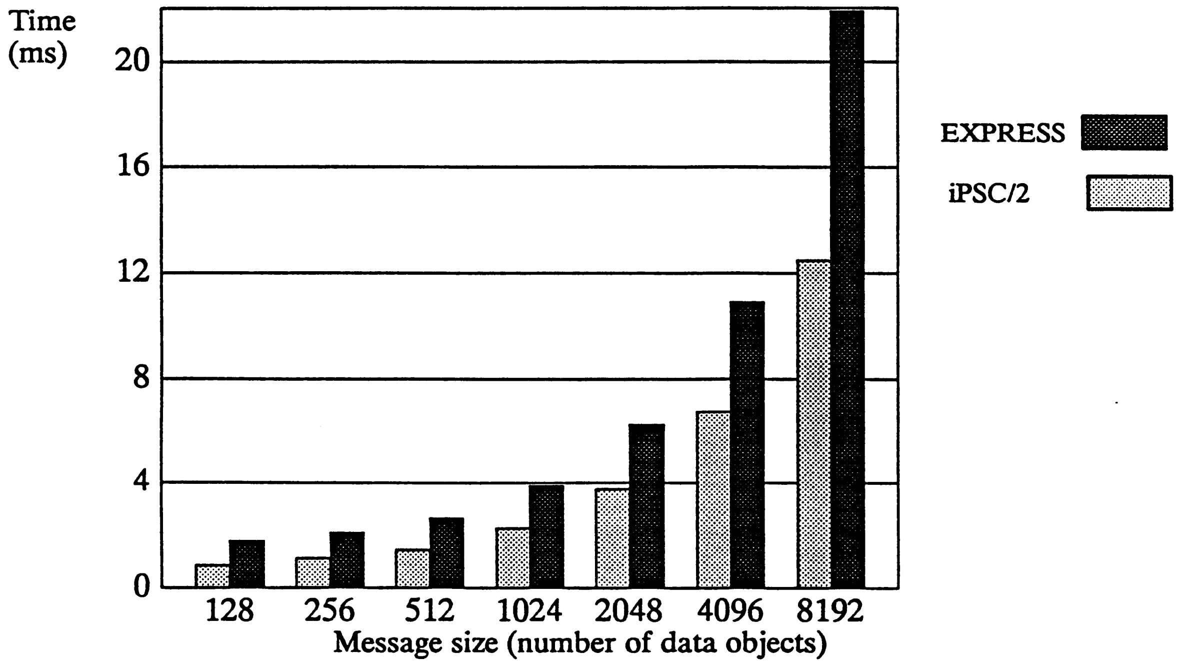

Figure 3: EXPRESS and iPSC/2 time for send operation versus message size

4.3 Measuring Communication Time between Nodes

In addition to measuring the execution time of CSEND and CRECV primitives for iPSC/2, and KWRIT and KXREAD primitives for EXPRESS, we used these primitives to measure the communication time. The communication time is measured by establishing two way communication between two nodes, that is, one node sends a message to the other node and waits for the reply. The size of the reply message is also the same as the send message. Synchronization is done before the start of message passing. This method has been recommended as a reliable mean of measuring communication time [5]. The communication time is obtained simply by dividing the total time (time to send and receive) by two.

It is worth mentioning that iPSC/2 uses different communication protocols for short and long messages. The communication time T for large messages, with L bytes, can be modeled by the following formula.

$$T = T_S + T_B L$$

where T_S is the set up time and T_B is the transmission time for one byte. The timing ratio of communication time obtained with EXPRESS to the communication time obtained with iPSC/2 is given by

$$\textit{Timing Ratio} = \frac{T_S(express) + T_B(express) \times L}{T_S(ipsc/2) + T_B(ipsc/2) \times L}$$

The communication times obtained by both EXPRESS and iPSC/2 primitives as well as the corresponding timing ratios are shown in Table 3. It is to be noted that the message size used in our experimentation is actually the number of words where each word consists of 4 bytes. In order to use the message size in the formula shown above, L should be obtained by multiplying the message size by 4. From the data shown in Table 3, the channel set up times and transmission times for both EXPRESS and iPSC/2, and the timing ratio can be summarized by as

$$\textit{Timing Ratio} = \frac{1.25 + 0.47 \times L}{0.73 + 0.37 \times L}$$

Timing results :

Table 3: Communication times for node to node communication (ms)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	0.720	0.44	1.63
128	1.497	0.92	1.64
256	1.705	1.09	1.57
512	2.213	1.48	1.49
1024	3.115	2.24	1.39
2048	5.045	3.76	1.34
4096	8.862	6.68	1.33
8192	16.570	12.50	1.33

The results shown in Table 3 indicate that the communication time obtained by using EXPRESS is greater than the communication time obtained by using iPSC/2 primitives by a factor of 1.636 to 1.326. The time ratio, however, shows a decreasing trend for larger messages. The communication times show in Table 2.2 are also plotted in Figure 4.

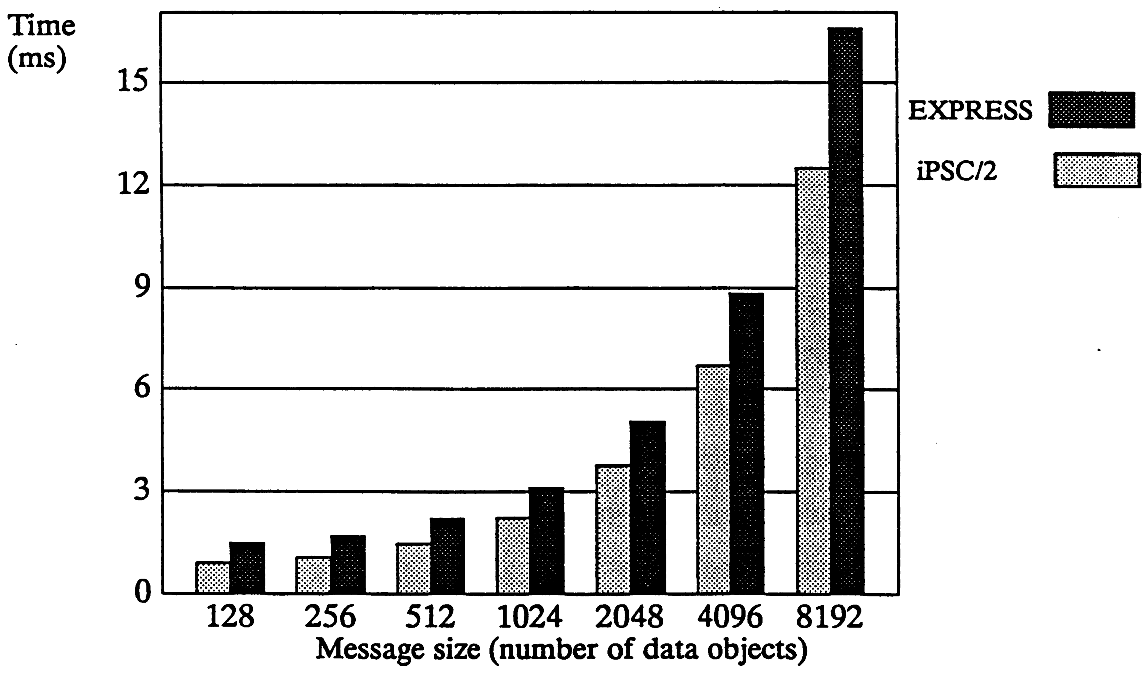

Figure 4: Communication Time using EXPRESS and iPSC/2 primitives

4.4 Host to Node Exchange Operations

The function of data exchange operation is to perform send and receive in one step, that is, the data is first sent and then received. It is also possible for one node to send and received data from different nodes. However, in our experiment, we considered send and receive with same one to one communication only. The send and receive used for exchange operation are of blocking nature and provide synchronization. An equivalent iPSC/2 primitives also exists for this kind of operation. Both primitives are described below.

EXPRESS Host to Node Exchange Routine :

```
INTEGER FUNCTION KXCHAN(IBUF, ISRC, ITYPE, OBUF, OLEN,
ODEST, OTYPE)
```

Equivalent iPSC/2 routine :

```
INTEGER FUNCTION CSENDRECV(TYPE, SBUF, SLEN, TONODE,
TOPID, TOTYPE, RBUF, RLEN)
```

Timing results :

Table 4 shows the timing results for exchange operation between host and node 0 for message size ranging from 10 to 8192.

Table 4: Timing results for the host to node exchange operation (ms)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	20.93	2.37	8.831
128	22.60	8.75	2.583
256	22.89	8.90	2.572
512	32.70	8.88	3.682
1024	36.13	10.78	3.352
2048	37.00	12.29	3.010
4096	37.79	18.80	2.010
8192	52.24	31.67	1.650

4.5 Node to Node Exchange Operations

The routines used for node–node exchange operation are the same as those used for host–node exchange operation.

Timing results :

Table 5 shows the timing results for exchange operation between node 0 and node 1 for message size ranging from 10 to 8192.

Table 5 : Timing results (ms) for exchange operation with source node 0 and destination node 1

	EXPRESS		iPSC/2		Timing Ratio	
Message Size	Node 0	Node 1	Node 0	Node 1	Node 0	Node 1
10	1.44	1.44	0.86	0.73	1.67	1.97
128	2.75	2.75	1.80	1.53	1.53	1.80
256	3.21	3.30	2.17	1.90	1.48	1.74
512	4.09	4.06	2.99	2.57	1.37	1.58
1024	5.09	5.15	4.46	4.06	1.14	1.27
2048	7.06	7.06	7.49	7.08	0.94	0.99
4096	10.89	10.91	13.25	12.96	0.82	0.84
8192	18.60	18.56	25.08	24.72	0.74	0.75

The relative performance of exchange operation clearly depends on the message size. For small messages, EXPRESS performs slowly as compared to iPSC/2. However, for long messages, the performance of EXPRESS is better than iPSC/2.

4.6 Broadcast from Host to Nodes

For data broadcast, EXPRESS provides a special routine, KXBROD, for sending the message from the host to all other nodes, and the host and nodes have to execute the same routine by specifying the host as source. On the other hand, iPSC/2 does not provide a special routine, rather CRECV and CSEND are executed by the receiving node(s) and host node respectively. Both routines are described below.

***EXPRESS* routine KXBROAD :**

INTEGER FUNCTION KXBROD(BUF, ORIGIN, LENGTH
NNODES, NODEL, TYPE)

Equivalent *iPSC/2* operation:

The equivalent iPSC/2 operation written for this purpose use the following routines:

SUBROUTINE CSEND(TYPE, BUF, LENGTH, -1, PID)
(at the origin node)
SUBROUTINE CRECV(TYPE, BUF, LENGTH)
(at all of the receiving nodes)

Timing results :

Host to node broadcast was timed for 4, 8, 16 and 32 nodes for message size varying from 10 to 2048. The results are tabulated in Tables 6.1 – 6.4. These results show the maximum of all nodes which are receiving the message from the host.

Table 6.1 : Node Time (ms) to receive data from host with broadcast (4 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	2.00	2.10	0.95
128	3.52	3.24	1.09
256	4.37	3.50	1.25
512	4.80	4.61	1.04
1024	7.04	6.66	1.06
2048	13.04	9.94	1.31

Table 6.2 : Node Time (ms) to receive data from host with broadcast (8 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	2.05	1.92	1.07
128	4.09	3.34	1.23
256	4.55	3.87	1.18
512	5.98	4.73	1.26
1024	8.91	7.17	1.24
2048	16.89	12.42	1.36

Table 6.3 : Node Time (ms) to receive data from host with broadcast (16 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	2.02	1.90	1.06
128	4.82	3.44	1.40
256	5.07	3.70	1.37
512	7.16	5.67	1.26
1024	10.81	7.87	1.37
2048	20.31	14.14	1.44

Table 6.4 : Node Time (ms) to receive data from host with broadcast (32 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	2.03	1.91	1.06
128	5.26	3.78	1.39
256	5.91	3.92	1.51
512	8.41	5.98	1.41
1024	12.75	8.95	1.43
2048	23.95	16.79	1.43

From the tables shown above, we notice that the performance of both systems shows consistent patterns. EXPRESS performs as good as iPSC/2 for smaller number of nodes but iPSC/2 performs better for larger number of nodes.

4.7 Broadcast From One Node to All Other Nodes

The same EXPRESS and iPSC/2 routines were used for one node to all node data broadcast which were used for host to nodes broadcast. The sending times at the single source node and the maximum of the receiving times at all of the receiving nodes were measured for various message sizes.

Timing results:

The timing results for both sending and receiving nodes are summarized in Tables 7.1, Table 7.2 and Table 7.3 for 4, 8 and 16 nodes, respectively.

Table 7.1 : Timing results (ms) for the node to nodes broadcast operation (4 NODES)

	EXPRESS		iPSC/2		Timing Ratio	
Message Size	send	recv.(max)	send	recv.(max)	send	recv.
10	0.67	1.58	0.42	0.63	1.60	2.51
128	1.54	2.94	1.22	1.64	1.26	1.79
256	1.89	3.45	1.57	2.01	1.20	1.72
512	2.56	4.39	2.25	2.75	1.14	1.60
1024	4.00	6.19	3.72	4.21	1.10	1.47
2048	6.96	10.10	6.62	7.14	1.06	1.42
4096	12.95	17.68	12.63	13.00	1.03	1.36
8192	24.78	33.25	24.26	24.74	1.02	1.34

Table 7.2 : Timing results (ms) for the node to nodes broadcast operation (8 NODES)

	EXPRESS		iPSC/2		Timing Ratio	
Message Size	send	recv.(max)	send	recv.(max)	send	recv.
10	0.94	2.17	0.47	0.72	2.00	3.13
128	2.18	4.24	1.62	2.28	1.35	1.86
256	2.73	5.02	2.17	2.82	1.26	1.78
512	3.78	6.47	3.29	3.90	1.15	1.66
1024	5.86	9.24	5.42	6.09	1.10	1.52
2048	10.35	14.97	9.83	10.56	1.11	1.42
4096	19.35	26.48	18.67	19.29	1.04	1.37
8192	37.02	49.85	36.24	36.89	1.02	1.35

Table 7.3 : Timing results (ms) for the node to nodes broadcast operation (16 NODES)

	EXPRESS		iPSC/2		Timing Ratio	
Message Size	send	recv.(max)	send	recv.(max)	send	recv.
10	1.18	2.74	0.59	0.86	2.00	3.19
128	2.86	5.58	2.10	2.91	1.36	1.92
256	3.58	6.66	2.82	3.64	1.27	1.83
512	5.02	8.56	4.29	5.13	1.17	1.67
1024	7.78	12.29	7.14	7.99	1.09	1.54
2048	13.78	19.95	13.01	13.87	1.06	1.44
4096	25.78	35.19	24.79	25.59	1.04	1.38
8192	49.76	66.37	48.77	49.01	1.01	1.35

Table 7.4 : Timing results (ms) for the node to nodes broadcast operation (32 NODES)

	EXPRESS		iPSC/2		Timing Ratio	
Message Size	send	recv.(max)	send	recv.(max)	send	recv.
10	1.53	3.39	0.63	0.98	2.000	3.19
128	3.57	6.93	2.55	3.51	1.400	1.97
256	4.46	8.17	3.49	4.39	1.270	1.83
512	6.22	10.73	5.32	6.28	1.170	1.67
1024	9.71	15.34	8.91	9.90	1.090	1.54
2048	17.15	24.90	16.25	17.30	1.060	1.44
4096	32.15	44.19	30.92	31.87	1.040	1.38
8192	61.55	83.49	60.25	61.14	1.014	1.35

The timing values for node to nodes broadcast indicate that EXPRESS and iPSC/2 perform almost identically for the sending node whereas receiving time with EXPRESS is higher. The difference in receiving time of both systems is higher for small messages but becomes less for larger messages. The same observation holds when broadcast is done to 4, 8, 16 or 32 nodes.

4.8 Global Reduction Operations

Global operations for producing reduction across all or a set of processors are frequently used in parallel programs. EXPRESS provides only one routine to perform any kind of global reduction operation and the user has to write its own function which is given as argument FUNC to the reduction routine. On the other hand, iPSC/2 provides separate routines for each operation. The global routines provided by iPSC/2 include global OR, AND, EXCLUSIVE OR , sum, maximum, minimum, etc. However, in our study, only global sum, global product and global maximum were tested. EXPRESS has extra facility of specifying the number of nodes and node id's which need to participate in global operation. The format of these routines is as follows.

EXPRESS global reduction operations:

```
INTEGER FUNCTION KXCOMB(BUF, FUNC, SIZE, ITEMS
                              NNODES, NODEL, TYPE)
```

Equivalent *iPSC/2* routine:

```
SUBROUTINE GxSUM(BUF, LENGTH, WORK)
SUBROUTINE GxPROD(BUF, LENGTH, WORK)
```

SUBROUTINE GxHIGH(BUF, LENGTH, WORK)

Note: x in GxOP stands for the type of data which can be I, S or D, for integer, real and double precision, respectively. It has already been mentioned that all of our tests were carried out with real data type.

Timing results:

We obtained timing results for global addition, global multiplication and global maximum. For EXPRESS, these three operations had to be written separately. For all three operations, data type used was 4 bytes floating point and the array size in each process was varied from 10 to 256.

Note: Tests for global operations for array size greater than 256 (in each node) cannot be carried out under EXPRESS.

Timing results for global sum:

The results for global sum are shown in Table 8.1 to Table 8.4, for 4, 8, 16 and 32 nodes, respectively.

Table 8.1 : Timing results (ms) for the global sum operation (4 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	1.077	0.967	1.11
64	1.219	1.092	1.12
128	1.281	1.153	1.11
256	1.378	1.239	1.11

Table 8.2 : Timing results (ms) for the global sum operation (8 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	1.151	1.016	1.13
64	1.291	1.168	1.10
128	1.361	1.235	1.10
256	1.442	1.323	1.09

Table 8.3 : Timing results (ms) for the global sum operation (16 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	1.206	1.062	1.14
64	1.347	1.234	1.09
128	1.410	1.292	1.09
256	1.506	1.384	1.08

Table 8.4 : Timing results (ms) for the global sum operation (32 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	1.249	1.107	1.13
64	1.393	1.279	1.09
128	1.453	1.342	1.08
256	1.540	1.429	1.08

Timing results for global multiplication:

The results for global multiplication are summarized in Table 8.5 to Table 8.8, for 4, 8, 16 and 32 nodes, respectively.

Table 8.5 : Timing results (ms) for the global multiplication operation (4 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	1.076	0.970	1.11
64	1.222	1.131	1.08
128	1.284	1.192	1.08
256	1.380	1.283	1.08

Table 8.6 : Timing results (ms) for the global multiplication operation (8 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	1.158	1.033	1.12
64	1.293	1.198	1.08
128	1.365	1.269	1.08
256	1.448	1.372	1.07

Table 8.7 : Timing results (ms) for the global multiplication operation (16 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	1.223	1.084	1.13
64	1.352	1.266	1.07
128	1.413	1.336	1.06
256	1.508	1.422	1.06

Table 8.8 : Timing results (ms) for the global multiplication operation (32 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	1.252	1.131	1.11
64	1.395	1.315	1.07
128	1.461	1.391	1.05
256	1.546	1.471	1.05

Timing results for global maximum:

The results for global maximum are summarized in Table 8.9 to Table 8.12, for 4, 8, 16 and 32 nodes, respectively.

Table 8.9 : Timing results (ms) for the global maximum operation (4 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	1.076	0.968	1.11
64	1.242	1.097	1.12
128	1.278	1.153	1.10
256	1.366	1.248	1.10

Table 8.10 : Timing results (ms) for the global maximum operation (8 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	1.150	1.013	1.13
64	1.285	1.175	1.10
128	1.351	1.250	1.08
256	1.433	1.321	1.08

Table 8.11 : Timing results (ms) for the global maximum operation (16 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	1.209	1.068	1.13
64	1.344	1.244	1.08
128	1.405	1.303	1.08
256	1.497	1.389	1.08

Table 8.12 : Timing results (ms) for the global maximum operation (32 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	1.298	1.125	1.15
64	1.389	1.296	1.08
128	1.446	1.353	1.07
256	1.531	1.428	1.07

Timing results for global reduction operation, addition, multiplication and maximum, indicate that for all three global operations, EXPRESS performs 5 to 10 percent slower than iPSC/2 for large message sizes. For small array sizes, such as 10 words, EXPRESS performs 10 to 15 percenet slower than iPSC/2.

4.9 Global Concatenation Operations

Global concatenation performs packing chunks of arrays from the participating nodes into one array. Each participating processor receives the resultant array. This primitive can also be used for all–to–all broadcasting.

EXPRESS provides the option to perform concatenation across all or a set of processors whereas with iPSC/2 all nodes have to call the concatenation routine. EXPRESS has extra facility of specifying the number of nodes and node id's which need to participate in global operation. This primitive also leaves into an array 'SIZES' the contribution made by each processor in term of array size. The format of these routines is as follows.

EXPRESS global concatenation operation :

```
INTEGER FUNCTION KXCONC(MYBUF, MYBYTE, RESBUF, RESIZE,
                        SIZES, NNODES, NODEL, TYPE)
```

Equivalent iPSC/2 routine :

SUBROUTINE GCOLX(X, XLENGTHS, RES)

Timing results :

Table 9.1 to Table 9.4 show the timing results for global concatenation operation for 4, 8, 16 and 32 nodes for array size 10 to 2048. Note that maximum array size used in our experiments is 2048 words. Since this is the array size in each node, the maximum size of the resultant array is thus 64k, when 32 processors are used.

Table 9.1: Timing results (ms) for the concatenation operation (4 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	2.32	1.32	1.76
64	4.68	2.93	1.60
128	5.11	3.77	1.36
256	6.22	5.56	1.06
512	9.25	8.73	1.06
1024	16.85	12.96	1.30
2048	32.23	21.86	1.47

Table 9.2: Timing results (ms) for the concatenation operation (8 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	5.28	2.39	2.21
64	11.57	4.32	2.68
128	12.56	5.66	2.22
256	15.36	8.77	1.75
512	21.32	14.96	1.43
1024	33.82	28.17	1.20
2048	61.28	55.82	1.10

Table 9.3: Timing results (ms) for the concatenation operation (16 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	11.35	3.67	3.09
64	25.3	6.27	4.04
200	27.60	9.00	3.07
256	33.19	14.61	2.27
512	46.79	27.35	1.71

1024	74.25	54.95	1.35
2048	130.40	113.52	1.24

Table 9.4: Timing results (ms) for the concatenation operation (32 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	23.10	5.06	4.57
100	53.16	9.69	4.30
200	59.56	15.61	3.09
256	70.80	28.17	2.51
512	98.54	53.78	1.78
1024	158.04	111.14	1.42
2048	278.04	223.20	1.25

4.10 Vector Send/Recv. Operations

Vector read and vector write routines are used to reading and writing vectors with an additional facility of transmitting non-contiguous blocks of data. These routine can be called from host and node processors. At the sending node, ITEMS objects each of size SIZE bytes separated by OFFSET are transmitted. On the receiving end, the received data is read into memory blocks separated by OFFSET. However, the basic functionality of vector read and vector write is same as KXREAD and KXWRIT. iPSC/2 does not provide such primitives. To make a fair comparison, we performed array packing with CSEND and array unpacking with CRECV at the sending and receiving nodes, respectively. The implementation includes the option of packing and unpacking at either ends.

EXPRESS Vector Send/Recv routine :

INTEGER FUNCTION KXVREAD(IBUF, SIZE, OFFSET, ITEMS, SRC, TYPE)

INTEGER FUNCTION KXVWRI(IBUF, SIZE, OFFSET, ITEMS, DEST, TYPE)

Equivalent iPSC/2 routine :

The equivalent iPSC/2 operation written for this purpose uses the following routines along with extra array packing and unpacking.

SUBROUTINE CRECV(TYPE, BUF, LENGTH)

SUBROUTINE CSEND(TYPE, BUF, LENGTH, -1, PID)

Timing results :

Table 10 shows the timing results for send/receive operation between node 0 and node 1 for message size ranging from 10 to 8192. The value of the offset at both the sending and receiving end was selected as 2.

Table 10 : Timing results(ms) for the vector send/receive operation (offset = 2 at source and destination)

	EXPRESS		iPSC/2		Timing Ratio	
Message Size	send	recv.	send	recv.	send	recv.
10	1.370	3.120	0.390	0.720	3.51	4.33
128	3.210	6.370	1.500	2.600	2.14	2.45
256	5.200	9.840	2.440	4.220	2.13	2.33
512	9.810	14.780	4.250	7.430	2.31	1.99
1024	19.130	24.600	7.870	13.910	2.43	1.77
2048	37.610	44.250	15.190	26.680	2.48	1.66
4096	75.340	83.600	29.890	52.340	2.52	1.60
8192	150.730	162.330	59.230	103.760	2.56	1.56

From Table 10, we observe that the performance of EXPRESS primitive for vector exchange is very poor. In spite of the fact that the execution time for iPSC/2 primitive includes the array packing and unpacking time, it is still less than the execution time of EXPRESS primitive.

4.11 Vector Exchange Operation

The vector operation for send/receive between any two nodes, as described above in section 4.10, can be accomplished in a bidirectional manner. This way two nodes can exchange vectors by using a single routine. For EXPRESS, this can be done using EXPRESS primitive described below. For iPSC/2, no such primitive exists and, therefore, CSENDRECV is used for two way data exchange and array packing is done at both ends to make comparison with the equivalent EXPRESS routine.

EXPRESS Exchange routine :

INTEGER FUNCTION KXVCHAN(IBUF, ISRC, ITYPE, OBUF, OLEN, ODEST, OTYPE)

Equivalent iPSC/2 operation :

The equivalent iPSC/2 operation written for this purpose uses the following routine along with extra array packing and unpacking option.

INTEGER FUNCTION CSENDRECV(TYPE, SBUF, SLEN, TONODE, TOPID, TOTYPE, RBUF, RLEN)

Timing results :

Timing results for two way vector exchange operation between node 0 and node 1 for message size ranging from 10 to 512, are given in Table 11.

Table 11: Timing results for the vector exchange operation (no offset at source and destination)

	EXPRESS		iPSC/2		Timing Ratio	
Message Size	send	recv.	send	recv.	send	recv.
10	4.13	4.15	1.01	0.94	4.09	4.42
64	39.34	39.46	2.35	2.18	16.79	18.10
128	136.79	136.77	3.315	3.025	41.26	45.21
256	509.20	509.23	5.14	4.84	99.07	105.21
512	1958.00	1957.50	8.85	8.53	229.54	229.48

The EXPRESS routine KXVCHAN is extremely slow, as shown in Table 10. The use of this routine is not recommended.

5. The Performance Comparison of Gaussian Elimination by using EXPRESS and iPSC/2

In order to compare the performance of the two programming model with real applications, we wrote and implemented a Gaussian Elimination algorithm with partial pivoting. The algorithm is based on row partitioning of the coefficient matrix. The algorithm was implemented with two programs written under iPSC/2 primitives and EXPRESS primitives. The main difference between two programs lies in the use of communication primitives like broadcast and global reduction operations such as performing the global addition and finding the global maximum. The execution time of both programs versus various matrix sizes using 8 nodes are shown in Figure 5. Figure

6 and Figure 7 show the execution time of Gaussian Elimination for 16 and 32 nodes, respectively.

For 8 processors, the difference in execution times of EXPRESS and iPSC/2 is 1 to 2 seconds. However, EXPRESS gets relatively slower for larger number of processors. The execution time, with larger number of processors is higher for the EXPRESS version. This difference becomes 2 to 3 seconds if 16 processors are used. For 32 processors, the difference becomes up to 10 seconds for a matrix size of 512. The EXPRESS primitives used in this program include KXCOMB, KXREAD and KXWRIT which are substituted for equivalent iPSC/2 primitives, GSSUM, CSEND and CRECV. The timing results indicate that these primitives of EXPRESS and iPSC/2 perform almost identical, with the exception of send operation where CSEND is faster than KXWRIT. However, the increased execution time of EXPRESS version of Gaussian Elimination program is due to the fact that this program uses broadcast communication in such a way that the origin of communication is not known to the receiving processors. This can be easily implemented with iPSC/2 primitive, CSEND and CRECV, by calling these routines only once. The EXPRESS broadcast primitive, KXBROD could not be used in this case because the format of the routine requires the receiving processors to know the id of the sending processor. As a result, KXREAD and KXWRIT have been used. Broadcast was carried out by making a call to these routines for each processor. As a results, the execution time increases if this technique is used for large number of processors.

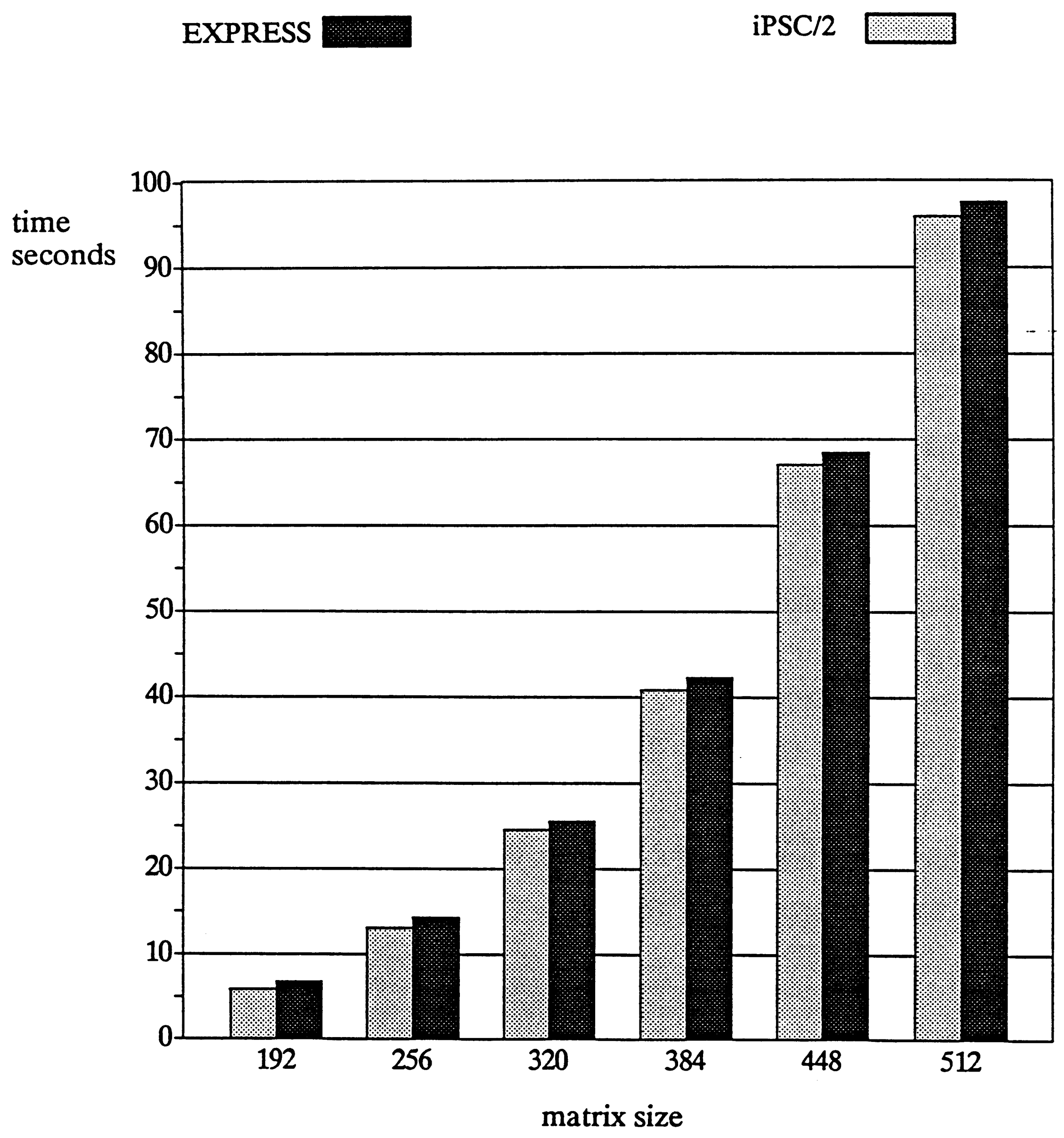

Figure 5: Performance of Gaussian Elimination on 8 Node Hypercube

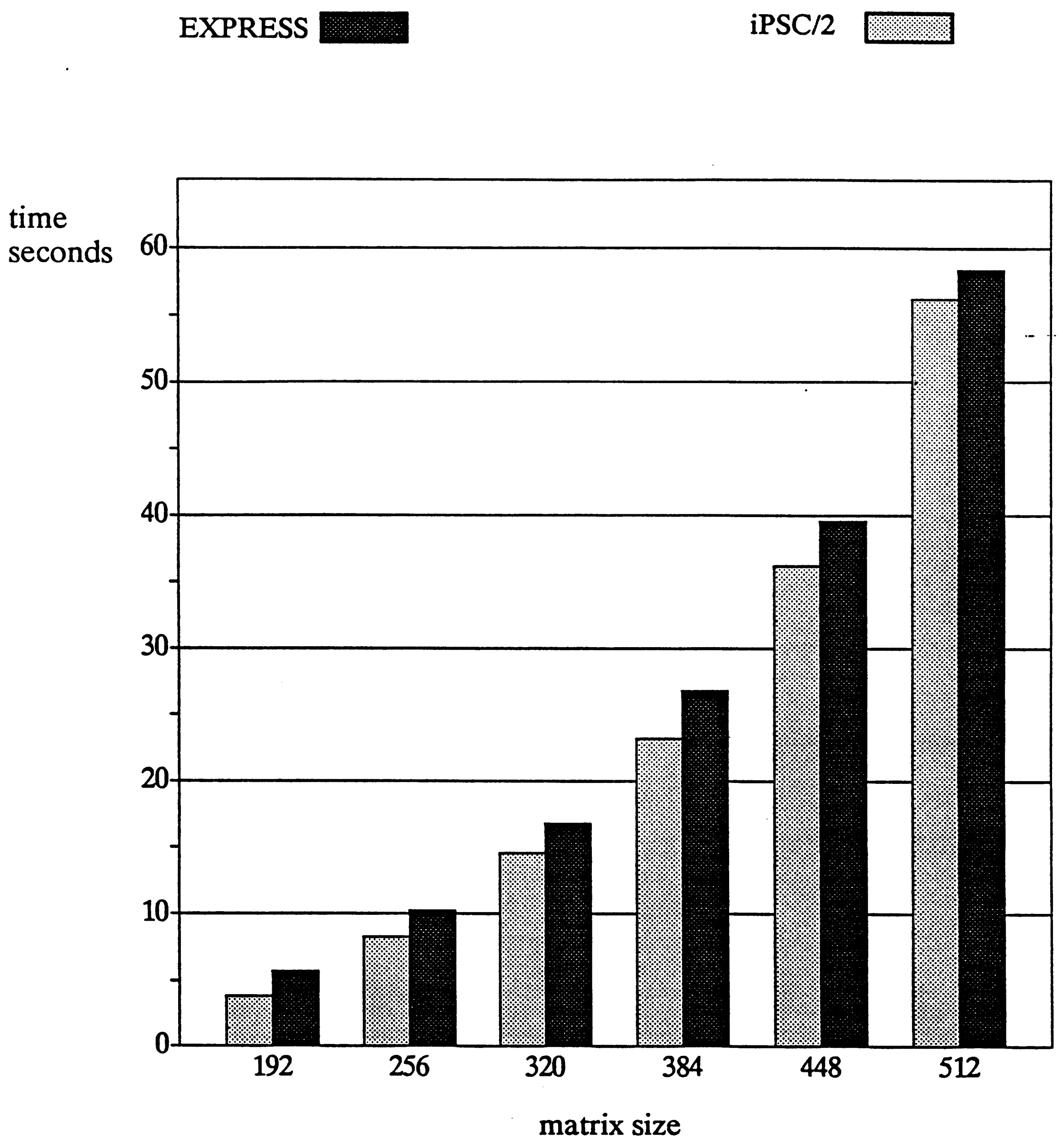

Figure 6: Performance of Gaussian Elimination on 16 Node Hypercube

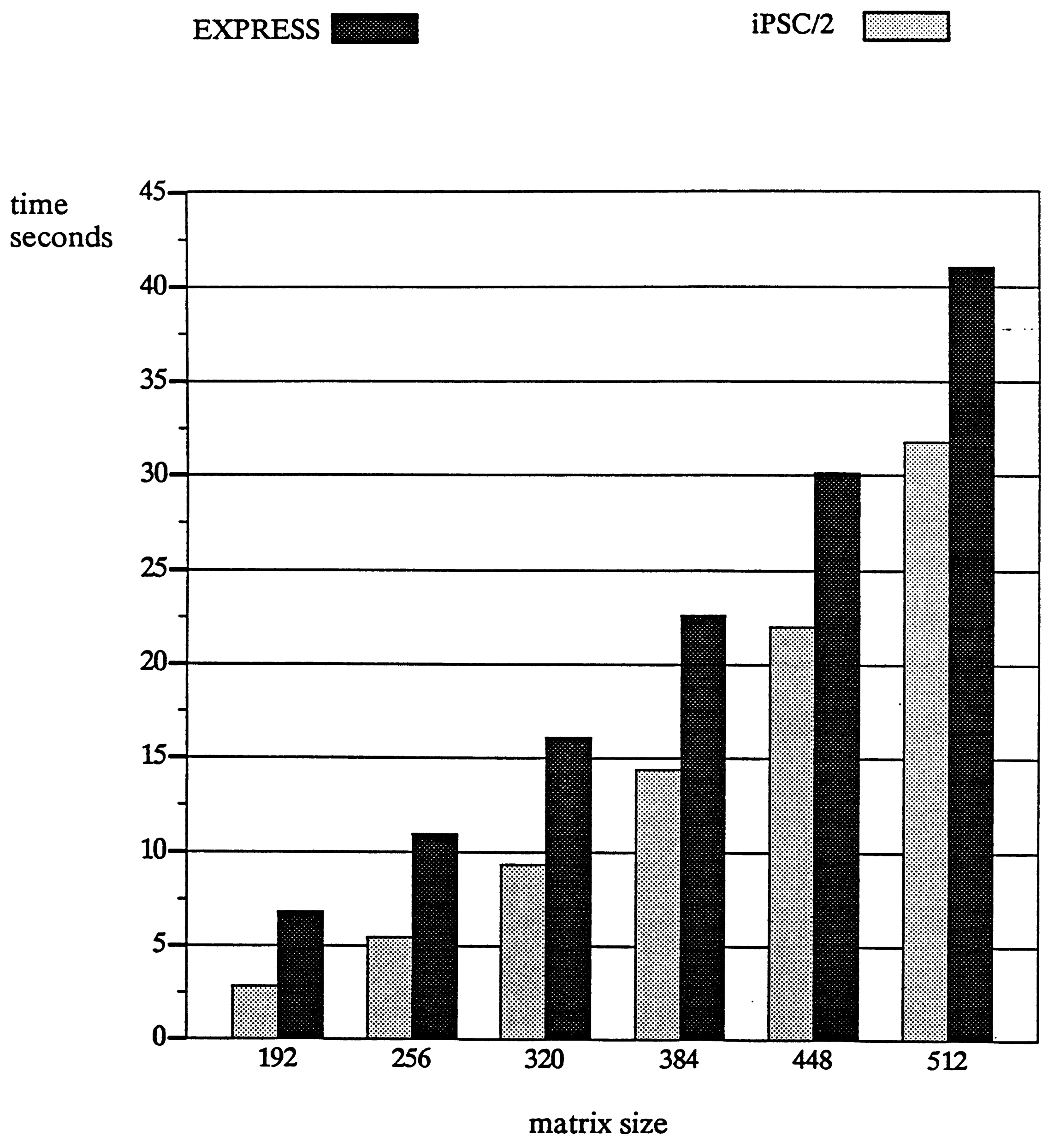

Figure 7: Performance of Gaussian Elimination on 32 Node Hypercube

6. Conclusions

In this report, we have presented a comparison of EXPRESS and iPSC/2 programming models. We have done this comparison by benchmarking various EXPRESS primitives as well as their equivalent iPSC/2 primitives. We have considered various environments, such as host–to–node and node–to–node communication, for testing the performance of these primitives and have analyzed the effect of data size on overall timing. For making a fair comparison, by writing new programs, we also implemented some of the equivalent operations of EXPRESS, which are not provided by iPSC/2. We considered both short and long messages.

Vector functions are useful EXPRESS facilities which are not provided by iPSC/2. On the other hand, EXPRESS has not implemented non–blocking send and receive primitives on the hypercube system so far. The EXPRESS broadcast primitive KXBROD requires the receiving processors to know the id of the sending processor, whereas iPSC/2 primitive CRECV does not. In some cases, such as in the Gaussian Elimination program, the receiving processors do not know the source of broadcasting, and KXBROD cannot be used.

The benchmark comparison of two programming models reveals many insights which can be summarized as follows.

- The iPSC/2 primitives clearly outperform EXPRESS primitives in most of the cases. A few exceptions include the node to node exchange for large messages.
- However, in many cases, the performance of EXPRESS primitives is comparable with iPSC/2 primitives.
- The most important performance measure is the node to node communication time. The comparison shows EXPRESS primitives are 30% to 70% slower than iPSC/2 primitives.
- The receiving time of two models is comparable but the sending time of EXPRESS is higher.

EXPRESS provides a portable environment for parallel programming on different parallel machines with a rich set of utility functions. The cost of using EXPRESS is larger overhead. Larger overhead leads to worse performance, or it requires larger granularity. With the exception of a few cases, the timing ratio of EXPRESS and

iPSC/2 primitives is in the range of 1 to 2.5 (The EXPRESS vector exchange is extremely slow). However, we believe that is an implementation error). Assume communication time is 10% of total execution time, it causes less than 10% performance degradation. Considering its functionality, the overhead of EXPRESS appears not too large and affordable.

Acknowledgements

We thankfully acknowledge the support of Center for Research on Parallel Computation. This work was supported by a grant from National Science Foundation under Cooperative Agreement No. CCR–8809165.

References

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

[2] Express Fortran Reference Guide, Parasoft Corporation, 1990.

[3] iPSC/2 Fortran Language Reference Manual, Intel Corporation, 1989.

[4] I. Angus, G. Fox, J. Kim, D. Walker, *Solving Problems on Concurrent Processors, Volume II*, Prentice Hall, Englewood Cliffs, New Jersey, 1990.

[5] David Bradley, "First and Second Generation Hypercube Performance," Report No. UIUCDCS–R–88–1455, Department of Computer Science, University of Ilinois at Urbana–Champaign, Sept. 1988.

ꞓs for

ꞓoft [1], is a software programming environment
ꞓogeneous MIMD multiprocessors. It provides a
icating processes, mechanisms for data sharing,
erformance analyzing tools. An important feature
lities are carried out in a user transparent manner.
it an attractive tool set for developing parallel pro-
ures which allow an application to use appropriate
utomatic domain decomposition library which can
lem to the underlying topology of the parallel com-
valuation tools, using text and graphics, can be effec-
me performance of the program.

re of EXPRESS is that it is portable. This leads to two advantages. The first auvantage is that it can be implemented on a variety of machines such as NCUBE–1, NCUBE–2, Symult, and Intel iPSC/2 and iPSC/860 hypercubes, transputer arrays and shared memory BBN Butterfly system. In addition, it can be implemented on various types of workstation networks. The second advantage of portability is that programs written under EXPRESS for one machine can be run on another machine and the user does not have to worry about the hardware. The languages supported by EXPRESS are C and FORTRAN.

The communication primitives for message passing under EXPRESS make use of asynchronous communication which include exwrite and exread functions at the source and destination nodes, respectively. The EXPRESS kernel provides intermediate buffering and routing. A subset of Express environment is CrOS III which provides synchronous communication system [4]. The third subset of EXPRESS is CUBIX system which allows nodes to perform concurrent I/O.

As depicted in Figure 1, the EXPRESS environment consists of three layers. The lowest level consists of utilities for controlling the hardware. The medium level provides support for problem partitioning and communication between the nodes, and between the node and control processor. The highest level contains the facilities for node programs to perform I/0 to the host operating system.

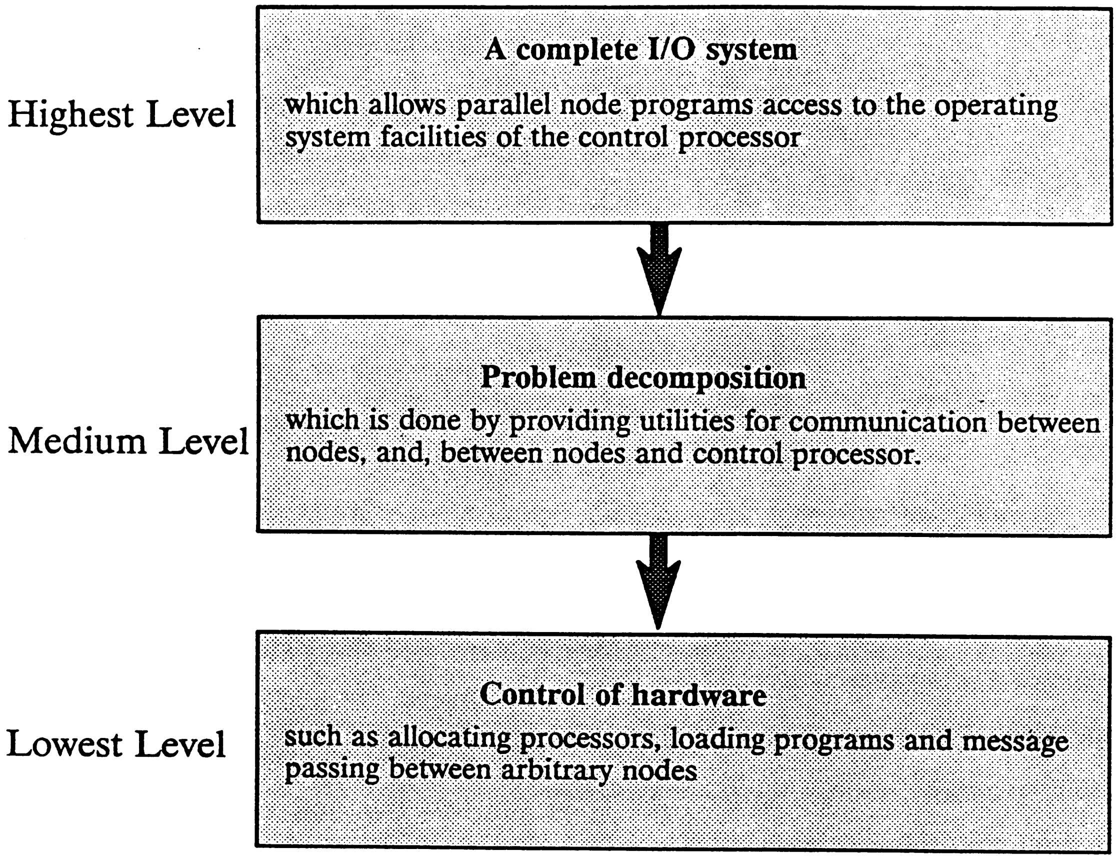
Highest Level
A complete I/O system
which allows parallel node programs access to the operating system facilities of the control processor
Medium Level
Problem decomposition
which is done by providing utilities for communication between nodes, and, between nodes and control processor.
Lowest Level
Control of hardware
such as allocating processors, loading programs and message passing between arbitrary nodes

Figure 1: Three layers of EXPRESS

2. The *EXPRESS* Programming Models

EXPRESS allows parallel programs to develop in the following four different styles.

(1) The conventional ***Host–Node*** Style in which the control part of the program appears in the host program and the computation intensive part appears in node programs.

(2) The ***Cubix*** environment in which only node program is written and it interfaces to the outside world through graphics and text server.

(3) The ***Profile*** environment which can be added to both Host–Node and Cubix programming models.

(4) The ***Plotix*** environment which provides extra graphics features to analyze program profiling.

3. The EXPRESS Utilities

EXPRESS provides a number of utilities which can be classified as follows.

(1) User Commands:

• Profiling Tools

I. ctool (Analyze communication profile data).

II. xtool (Analyze execution profile data).

III. etool (Analyze events profile data).

• System administration such as configuration, reboot and reload.

• User's usage of system resources.

• Source level debugger (communication, execution and events).

(2) Compilers:

C and Fortran Compilers for hypercubes.

C and Fortran Compilers for transputers.

(3) Initialization:

• Starting up *EXPRESS* system.

(4) Processor Control:

• Allocation and de–allocation of processors.

• Loading program into all nodes.

• Loading program into selected nodes.

(5) Interprocessor communication:

• Blocking communication among nodes e.g read and write a message, read and write a vector.

• Global communication.

e.g broadcast, synchronous data exchange, global concatenation, global reduction operations, synchronization, etc.

- Asynchronous communication.

  e.g non–blocking read and write.
- Hardware dependent communication.

  e.g reading and writing data on a channel.

(6) Decomposition Tool:

- Initializing decomposition system.
- Map user domain coordinates to processor number.
- Distributing data to processors.
- Distribute processors on user domain.
- Distributing data to processors.
- Determining run–time environment.

(7) I/O Routines

(8) Multi–tasking

(9) Multi–Hosting

(10) Graphics

(11) Utilities Routines

4. The Performance of Some EXPRESS and Equivalent iPSC/2 Routines

An important issue associated with programming tools is the execution speed of the basic primitives which are frequently used in programs. Examples of such primitives include communication between nodes, communication between host and nodes, global reduction operations, data broadcast, concatenation etc. We have evaluated the performance of some of the basic primitives of EXPRESS, running on iPSC/2. To make comparison with iPSC/2, we also benchmarked equivalent iPSC/2 primitives. In those cases where equivalent primitives did not exist, we wrote routines that can perform the same function.

Our benchmark model consists of two host programs – one for EXPRESS and one for iPSC/2, and one node program for each primitive. Separate node programs are written for EXPRESS and iPSC/2. A host program consists of a menu through which a selection can be made to test one particular primitive. After the selection has been made, additional information such as the number of nodes, source and destination for communication, the choice of host–node communication or node–node communication, the message size, must be provided. The host program selects the appropriate node program and loads it into the nodes. It passes the additional information to the node program and sets the appropriate environment. Each tested primitive is timed with a loop which repeats the same test up to 200 times. At the end of each loop step, all timing information is recorded and appropriate parameters are reset where necessary. Synchronization is done where it is deemed necessary. At the end of the loop, the timing of each test step is sent back to the host which performs statistical analysis on it by calculating the mean value and variance. If the variance is very high, he same test is repeated with a higher number of repetitions. The timing results, in-ding the mean and variance, for the host and node operations are stored in two dif-ent files.

Since the message passing protocols on iPSC/2 are different for large and small messages, our tests included both small and large messages. The size of large messages was varied from 128 to 8192 words and the size of small messages was kept fixed as 10 words. The data type of message objects is real type for all tests, which consists of 4 bytes. The actual message size, therefore, is message size multiplied by 4. All timing values are given in milli–seconds. To compare EXPRESS with iPSC/2 primitives, a timing ratio of EXPRESS primitive versus iPSC/2 primitive is calculated and provided in the corresponding table.

4.1 Send/Recv. between Host and Node

The communication between host and node is an important routine that is mostly used for sending initial data and other information from host to nodes and getting the results from nodes to host. It is also used for I/O between nodes and the file system, through the host. The communication can be blocking as well as non–blocking. We considered only the blocking type. The same routines can be used for communication between nodes and between nodes and host. Following are the EXPRESS and iPSC/2 routines for blocking communication.

EXPRESS routines :

INTEGER FUNCTION KXREAD(BUF, LENGTH, SRC, TYPE)
INTEGER FUNCTION KXWRITE(BUF, LENGTH, DEST, TYPE)

Equivalent iPSC/2 routines:

SUBROUTINE CRECV(TYPE, BUF, LENGTH)
SUBROUTINE CSEND(TYPE, BUF, LENGTH, DEST, PID)

Timing results :

Table 1 contains the execution time of EXPRESS KXREAD and KXWRITE and iPSC/2 CSEND and CRECV for host to node communication, for various message sizes. We measured timing for both way communication, that is, sending data from host to node and measuring node time and receiving data from node to host and measuring node's sending time.

Table 1: Node Time to receive and send data to host (ms)

	EXPRESS		iPSC/2		Timing Ratio	
Message Size	recv.	send	recv.	send	recv.	send
10	1.93	0.36	1.46	0.34	1.32	1.06
128	3.16	9.98	2.91	5.91	1.04	1.69
256	3.30	10.91	3.29	5.97	1.00	1.83
512	3.86	11.26	3.54	6.31	1.09	1.78
1024	4.93	11.93	4.18	7.24	1.18	1.65
2048	6.82	13.51	5.83	8.84	1.17	1.53
4096	11.16	16.97	9.08	12.01	1.23	1.41
8192	19.34	23.33	15.60	18.48	1.24	1.26

The results indicate that for both EXPRESS and iPSC/2, the time to send data from node to host is greater than the time to receive data from host to node. We also observe that iPSC/2 performs better by a maximum factor of 1.322 for receiving the data and by a factor of 1.827 for sending data.

4.2 Send/Recv. between Nodes

We used the same routines for node to node communication that were used for host to node communication.

Timing Results :

Again, we measured two way timing, that is, the sending time at the sending node and receiving time at the receiving node. Table 2 summarizes these results along with timing ratios for both receive and send. To present these results pictorially, plots of these values for receive and send are given in Figure 2 and Figure 3.

Table 2.1: Timing results (ms) for the node to node communication
(One to one communication)

	EXPRESS		iPSC/2		Timing Ratio	
Message Size	recv.	send	recv.	send	recv.	send
10	0.39	1.05	0.35	0.40	1.11	2.63
128	0.77	1.81	0.77	0.95	1.00	1.91
256	0.95	2.10	0.94	1.14	1.06	1.84
512	1.30	2.69	1.25	1.47	1.04	1.83
1024	2.01	3.89	1.96	2.30	1.03	1.69
2048	3.53	6.25	3.44	3.75	1.03	1.67
4096	6.50	10.92	6.33	6.76	1.03	1.62
8192	12.38	21.94	12.25	12.50	1.01	1.76

From Table 2.1 , and Figure 2 and Figure 3, we notice that the performance of both systems is almost the same for receiving data. However, iPSC/2 outperforms EXPRESS for sending data.

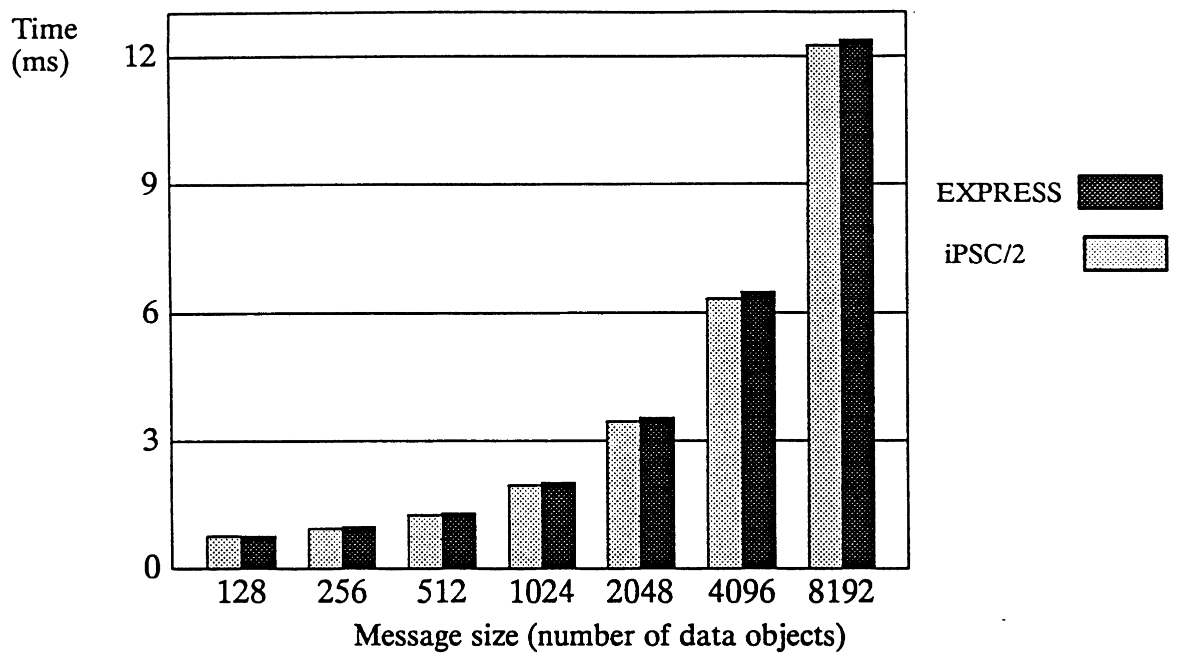

Figure 2: EXPRESS and iPSC/2 time for receive operation versus message size

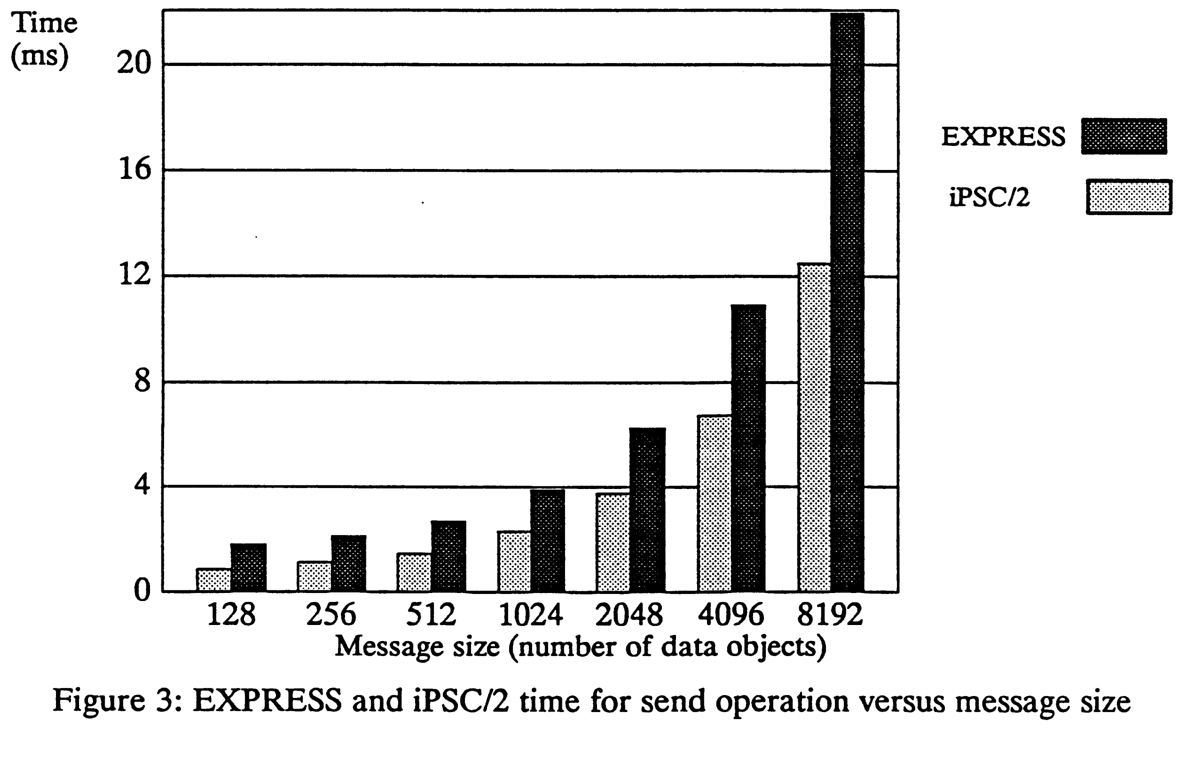

Figure 3: EXPRESS and iPSC/2 time for send operation versus message size

4.3 Measuring Communication Time between Nodes

In addition to measuring the execution time of CSEND and CRECV primitives for iPSC/2, and KWRIT and KXREAD primitives for EXPRESS, we used these primitives to measure the communication time. The communication time is measured by establishing two way communication between two nodes, that is, one node sends a message to the other node and waits for the reply. The size of the reply message is also the same as the send message. Synchronization is done before the start of message passing. This method has been recommended as a reliable mean of measuring communication time [5]. The communication time is obtained simply by dividing the total time (time to send and receive) by two.

It is worth mentioning that iPSC/2 uses different communication protocols for short and long messages. The communication time T for large messages, with L bytes, can be modeled by the following formula.

$$T = T_S + T_B L$$

where T_S is the set up time and T_B is the transmission time for one byte. The timing ratio of communication time obtained with EXPRESS to the communication time obtained with iPSC/2 is given by

$$Timing\ Ratio = \frac{T_S(express) + T_B(express) \times L}{T_S(ipsc/2) + T_B(ipsc/2) \times L}$$

The communication times obtained by both EXPRESS and iPSC/2 primitives as well as the corresponding timing ratios are shown in Table 3. It is to be noted that the message size used in our experimentation is actually the number of words where each word consists of 4 bytes. In order to use the message size in the formula shown above, L should be obtained by multiplying the message size by 4. From the data shown in Table 3, the channel set up times and transmission times for both EXPRESS and iPSC/2, and the timing ratio can be summarized by as

$$Timing\ Ratio = \frac{1.25 + 0.47 \times L}{0.73 + 0.37 \times L}$$

Timing results :

Table 3: Communication times for node to node communication (ms)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	0.720	0.44	1.63
128	1.497	0.92	1.64
256	1.705	1.09	1.57
512	2.213	1.48	1.49
1024	3.115	2.24	1.39
2048	5.045	3.76	1.34
4096	8.862	6.68	1.33
8192	16.570	12.50	1.33

The results shown in Table 3 indicate that the communication time obtained by using EXPRESS is greater than the communication time obtained by using iPSC/2 primitives by a factor of 1.636 to 1.326. The time ratio, however, shows a decreasing trend for larger messages. The communication times show in Table 2.2 are also plotted in Figure 4.

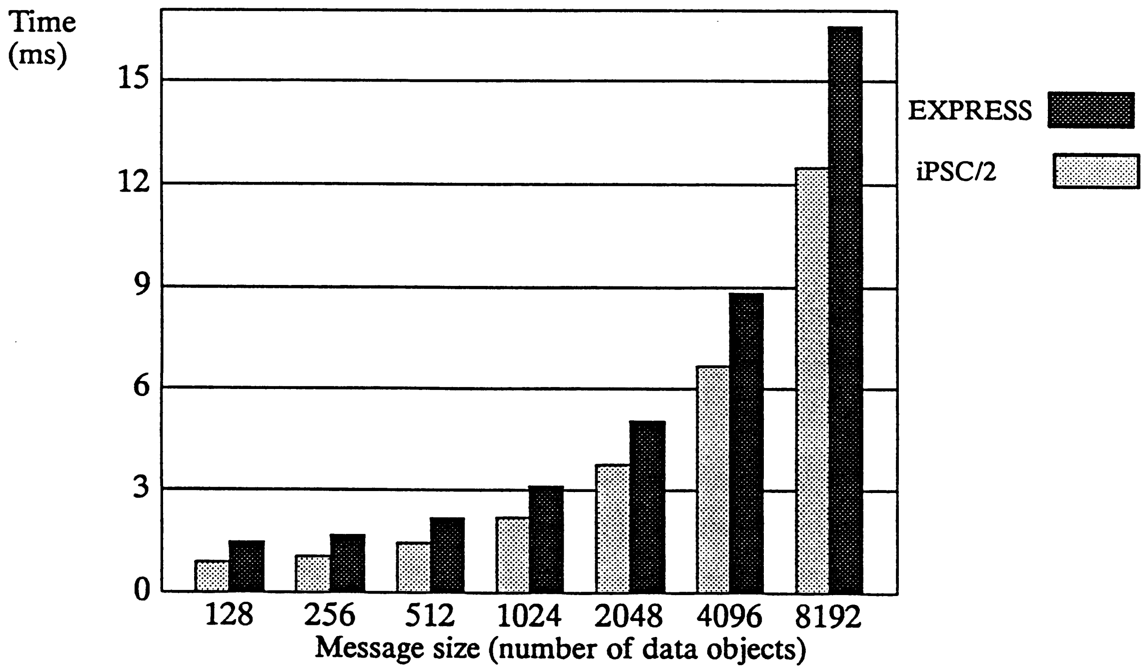

Figure 4: Communication Time using EXPRESS and iPSC/2 primitives

4.4 Host to Node Exchange Operations

The function of data exchange operation is to perform send and receive in one step, that is, the data is first sent and then received. It is also possible for one node to send and received data from different nodes. However, in our experiment, we considered send and receive with same one to one communication only. The send and receive used for exchange operation are of blocking nature and provide synchronization. An equivalent iPSC/2 primitives also exists for this kind of operation. Both primitives are described below.

<u>EXPRESS Host to Node Exchange Routine :</u>

INTEGER FUNCTION KXCHAN(IBUF, ISRC, ITYPE, OBUF, OLEN, ODEST, OTYPE)

<u>Equivalent iPSC/2 routine :</u>

INTEGER FUNCTION CSENDRECV(TYPE, SBUF, SLEN, TONODE, TOPID, TOTYPE, RBUF, RLEN)

<u>Timing results :</u>

Table 4 shows the timing results for exchange operation between host and node 0 for message size ranging from 10 to 8192.

Table 4: Timing results for the host to node exchange operation (ms)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	20.93	2.37	8.831
128	22.60	8.75	2.583
256	22.89	8.90	2.572
512	32.70	8.88	3.682
1024	36.13	10.78	3.352
2048	37.00	12.29	3.010
4096	37.79	18.80	2.010
8192	52.24	31.67	1.650

4.5 Node to Node Exchange Operations

The routines used for node–node exchange operation are the same as those used for host–node exchange operation.

Timing results :

Table 5 shows the timing results for exchange operation between node 0 and node 1 for message size ranging from 10 to 8192.

Table 5 : Timing results (ms) for exchange operation with source node 0 and destination node 1

	EXPRESS		iPSC/2		Timing Ratio	
Message Size	Node 0	Node 1	Node 0	Node 1	Node 0	Node 1
10	1.44	1.44	0.86	0.73	1.67	1.97
128	2.75	2.75	1.80	1.53	1.53	1.80
256	3.21	3.30	2.17	1.90	1.48	1.74
512	4.09	4.06	2.99	2.57	1.37	1.58
1024	5.09	5.15	4.46	4.06	1.14	1.27
2048	7.06	7.06	7.49	7.08	0.94	0.99
4096	10.89	10.91	13.25	12.96	0.82	0.84
8192	18.60	18.56	25.08	24.72	0.74	0.75

The relative performance of exchange operation clearly depends on the message size. For small messages, EXPRESS performs slowly as compared to iPSC/2. However, for long messages, the performance of EXPRESS is better than iPSC/2.

4.6 Broadcast from Host to Nodes

For data broadcast, EXPRESS provides a special routine, KXBROD, for sending the message from the host to all other nodes, and the host and nodes have to execute the same routine by specifying the host as source. On the other hand, iPSC/2 does not provide a special routine, rather CRECV and CSEND are executed by the receiving node(s) and host node respectively. Both routines are described below.

***EXPRESS* routine KXBROAD :**

INTEGER FUNCTION KXBROD(BUF, ORIGIN, LENGTH
NNODES, NODEL, TYPE)

Equivalent *iPSC/2* operation:

The equivalent iPSC/2 operation written for this purpose use the following routines:

SUBROUTINE CSEND(TYPE, BUF, LENGTH, -1, PID)
(at the origin node)
SUBROUTINE CRECV(TYPE, BUF, LENGTH)
(at all of the receiving nodes)

Timing results :

Host to node broadcast was timed for 4, 8, 16 and 32 nodes for message size varying from 10 to 2048. The results are tabulated in Tables 6.1 – 6.4. These results show the maximum of all nodes which are receiving the message from the host.

Table 6.1 : Node Time (ms) to receive data from host with broadcast (4 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	2.00	2.10	0.95
128	3.52	3.24	1.09
256	4.37	3.50	1.25
512	4.80	4.61	1.04
1024	7.04	6.66	1.06
2048	13.04	9.94	1.31

Table 6.2 : Node Time (ms) to receive data from host with broadcast (8 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	2.05	1.92	1.07
128	4.09	3.34	1.23
256	4.55	3.87	1.18
512	5.98	4.73	1.26
1024	8.91	7.17	1.24
2048	16.89	12.42	1.36

Table 6.3 : Node Time (ms) to receive data from host with broadcast (16 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	2.02	1.90	1.06
128	4.82	3.44	1.40
256	5.07	3.70	1.37
512	7.16	5.67	1.26
1024	10.81	7.87	1.37
2048	20.31	14.14	1.44

Table 6.4 : Node Time (ms) to receive data from host with broadcast (32 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	2.03	1.91	1.06
128	5.26	3.78	1.39
256	5.91	3.92	1.51
512	8.41	5.98	1.41
1024	12.75	8.95	1.43
2048	23.95	16.79	1.43

From the tables shown above, we notice that the performance of both systems shows consistent patterns. EXPRESS performs as good as iPSC/2 for smaller number of nodes but iPSC/2 performs better for larger number of nodes.

4.7 Broadcast From One Node to All Other Nodes

The same EXPRESS and iPSC/2 routines were used for one node to all node data broadcast which were used for host to nodes broadcast. The sending times at the single source node and the maximum of the receiving times at all of the receiving nodes were measured for various message sizes.

Timing results:

The timing results for both sending and receiving nodes are summarized in Tables 7.1, Table 7.2 and Table 7.3 for 4, 8 and 16 nodes, respectively.

Table 7.1 : Timing results (ms) for the node to nodes broadcast operation (4 NODES)

	EXPRESS		iPSC/2		Timing Ratio	
Message Size	send	recv.(max)	send	recv.(max)	send	recv.
10	0.67	1.58	0.42	0.63	1.60	2.51
128	1.54	2.94	1.22	1.64	1.26	1.79
256	1.89	3.45	1.57	2.01	1.20	1.72
512	2.56	4.39	2.25	2.75	1.14	1.60
1024	4.00	6.19	3.72	4.21	1.10	1.47
2048	6.96	10.10	6.62	7.14	1.06	1.42
4096	12.95	17.68	12.63	13.00	1.03	1.36
8192	24.78	33.25	24.26	24.74	1.02	1.34

Table 7.2 : Timing results (ms) for the node to nodes broadcast operation (8 NODES)

	EXPRESS		iPSC/2		Timing Ratio	
Message Size	send	recv.(max)	send	recv.(max)	send	recv.
10	0.94	2.17	0.47	0.72	2.00	3.13
128	2.18	4.24	1.62	2.28	1.35	1.86
256	2.73	5.02	2.17	2.82	1.26	1.78
512	3.78	6.47	3.29	3.90	1.15	1.66
1024	5.86	9.24	5.42	6.09	1.10	1.52
2048	10.35	14.97	9.83	10.56	1.11	1.42
4096	19.35	26.48	18.67	19.29	1.04	1.37
8192	37.02	49.85	36.24	36.89	1.02	1.35

Table 7.3 : Timing results (ms) for the node to nodes broadcast operation (16 NODES)

	EXPRESS		iPSC/2		Timing Ratio	
Message Size	send	recv.(max)	send	recv.(max)	send	recv.
10	1.18	2.74	0.59	0.86	2.00	3.19
128	2.86	5.58	2.10	2.91	1.36	1.92
256	3.58	6.66	2.82	3.64	1.27	1.83
512	5.02	8.56	4.29	5.13	1.17	1.67
1024	7.78	12.29	7.14	7.99	1.09	1.54
2048	13.78	19.95	13.01	13.87	1.06	1.44
4096	25.78	35.19	24.79	25.59	1.04	1.38
8192	49.76	66.37	48.77	49.01	1.01	1.35

Table 7.4 : Timing results (ms) for the node to nodes broadcast operation (32 NODES)

	EXPRESS		iPSC/2		Timing Ratio	
Message Size	send	recv.(max)	send	recv.(max)	send	recv.
10	1.53	3.39	0.63	0.98	2.000	3.19
128	3.57	6.93	2.55	3.51	1.400	1.97
256	4.46	8.17	3.49	4.39	1.270	1.83
512	6.22	10.73	5.32	6.28	1.170	1.67
1024	9.71	15.34	8.91	9.90	1.090	1.54
2048	17.15	24.90	16.25	17.30	1.060	1.44
4096	32.15	44.19	30.92	31.87	1.040	1.38
8192	61.55	83.49	60.25	61.14	1.014	1.35

The timing values for node to nodes broadcast indicate that EXPRESS and iPSC/2 perform almost identically for the sending node whereas receiving time with EXPRESS is higher. The difference in receiving time of both systems is higher for small messages but becomes less for larger messages. The same observation holds when broadcast is done to 4, 8, 16 or 32 nodes.

4.8 Global Reduction Operations

Global operations for producing reduction across all or a set of processors are frequently used in parallel programs. EXPRESS provides only one routine to perform any kind of global reduction operation and the user has to write its own function which is given as argument FUNC to the reduction routine. On the other hand, iPSC/2 provides separate routines for each operation. The global routines provided by iPSC/2 include global OR, AND, EXCLUSIVE OR , sum, maximum, minimum, etc. However, in our study, only global sum, global product and global maximum were tested. EXPRESS has extra facility of specifying the number of nodes and node id's which need to participate in global operation. The format of these routines is as follows.

***EXPRESS* global reduction operations:**

```
INTEGER FUNCTION KXCOMB(BUF, FUNC, SIZE, ITEMS
                                   NNODES, NODEL, TYPE)
```

Equivalent *iPSC/2* routine:

```
SUBROUTINE GxSUM(BUF, LENGTH, WORK)
SUBROUTINE GxPROD(BUF, LENGTH, WORK)
```

SUBROUTINE GxHIGH(BUF, LENGTH, WORK)

Note: x in GxOP stands for the type of data which can be I, S or D, for integer, real and double precision, respectively. It has already been mentioned that all of our tests were carried out with real data type.

Timing results:

We obtained timing results for global addition, global multiplication and global maximum. For EXPRESS, these three operations had to be written separately. For all three operations, data type used was 4 bytes floating point and the array size in each process was varied from 10 to 256.

Note: Tests for global operations for array size greater than 256 (in each node) cannot be carried out under EXPRESS.

Timing results for global sum:

The results for global sum are shown in Table 8.1 to Table 8.4, for 4, 8, 16 and 32 nodes, respectively.

Table 8.1 : Timing results (ms) for the global sum operation (4 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	1.077	0.967	1.11
64	1.219	1.092	1.12
128	1.281	1.153	1.11
256	1.378	1.239	1.11

Table 8.2 : Timing results (ms) for the global sum operation (8 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	1.151	1.016	1.13
64	1.291	1.168	1.10
128	1.361	1.235	1.10
256	1.442	1.323	1.09

Table 8.3 : Timing results (ms) for the global sum operation (16 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	1.206	1.062	1.14
64	1.347	1.234	1.09
128	1.410	1.292	1.09
256	1.506	1.384	1.08

Table 8.4 : Timing results (ms) for the global sum operation (32 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	1.249	1.107	1.13
64	1.393	1.279	1.09
128	1.453	1.342	1.08
256	1.540	1.429	1.08

Timing results for global multiplication:

The results for global multiplication are summarized in Table 8.5 to Table 8.8, for 4, 8, 16 and 32 nodes, respectively.

Table 8.5 : Timing results (ms) for the global multiplication operation (4 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	1.076	0.970	1.11
64	1.222	1.131	1.08
128	1.284	1.192	1.08
256	1.380	1.283	1.08

Table 8.6 : Timing results (ms) for the global multiplication operation (8 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	1.158	1.033	1.12
64	1.293	1.198	1.08
128	1.365	1.269	1.08
256	1.448	1.372	1.07

Table 8.7 : Timing results (ms) for the global multiplication operation (16 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	1.223	1.084	1.13
64	1.352	1.266	1.07
128	1.413	1.336	1.06
256	1.508	1.422	1.06

Table 8.8 : Timing results (ms) for the global multiplication operation (32 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	1.252	1.131	1.11
64	1.395	1.315	1.07
128	1.461	1.391	1.05
256	1.546	1.471	1.05

Timing results for global maximum:

The results for global maximum are summarized in Table 8.9 to Table 8.12, for 4, 8, 16 and 32 nodes, respectively.

Table 8.9 : Timing results (ms) for the global maximum operation (4 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	1.076	0.968	1.11
64	1.242	1.097	1.12
128	1.278	1.153	1.10
256	1.366	1.248	1.10

Table 8.10 : Timing results (ms) for the global maximum operation (8 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	1.150	1.013	1.13
64	1.285	1.175	1.10
128	1.351	1.250	1.08
256	1.433	1.321	1.08

Table 8.11 : Timing results (ms) for the global maximum operation (16 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	1.209	1.068	1.13
64	1.344	1.244	1.08
128	1.405	1.303	1.08
256	1.497	1.389	1.08

Table 8.12 : Timing results (ms) for the global maximum operation (32 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	1.298	1.125	1.15
64	1.389	1.296	1.08
128	1.446	1.353	1.07
256	1.531	1.428	1.07

Timing results for global reduction operation, addition, multiplication and maximum, indicate that for all three global operations, EXPRESS performs 5 to 10 percent slower than iPSC/2 for large message sizes. For small array sizes, such as 10 words, EXPRESS performs 10 to 15 percenet slower than iPSC/2.

4.9 Global Concatenation Operations

Global concatenation performs packing chunks of arrays from the participating nodes into one array. Each participating processor receives the resultant array. This primitive can also be used for all–to–all broadcasting.

EXPRESS provides the option to perform concatenation across all or a set of processors whereas with iPSC/2 all nodes have to call the concatenation routine. EXPRESS has extra facility of specifying the number of nodes and node id's which need to participate in global operation. This primitive also leaves into an array 'SIZES' the contribution made by each processor in term of array size. The format of these routines is as follows.

<u>**EXPRESS global concatenation operation :**</u>

```
INTEGER FUNCTION KXCONC(MYBUF, MYBYTE, RESBUF, RESIZE,
                        SIZES, NNODES, NODEL, TYPE)
```

Equivalent iPSC/2 routine :

SUBROUTINE GCOLX(X, XLENGTHS, RES)

Timing results :

Table 9.1 to Table 9.4 show the timing results for global concatenation operation for 4, 8, 16 and 32 nodes for array size 10 to 2048. Note that maximum array size used in our experiments is 2048 words. Since this is the array size in each node, the maximum size of the resultant array is thus 64k, when 32 processors are used.

Table 9.1: Timing results (ms) for the concatenation operation (4 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	2.32	1.32	1.76
64	4.68	2.93	1.60
128	5.11	3.77	1.36
256	6.22	5.56	1.06
512	9.25	8.73	1.06
1024	16.85	12.96	1.30
2048	32.23	21.86	1.47

Table 9.2: Timing results (ms) for the concatenation operation (8 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	5.28	2.39	2.21
64	11.57	4.32	2.68
128	12.56	5.66	2.22
256	15.36	8.77	1.75
512	21.32	14.96	1.43
1024	33.82	28.17	1.20
2048	61.28	55.82	1.10

Table 9.3: Timing results (ms) for the concatenation operation (16 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	11.35	3.67	3.09
64	25.3	6.27	4.04
200	27.60	9.00	3.07
256	33.19	14.61	2.27
512	46.79	27.35	1.71

1024	74.25	54.95	1.35
2048	130.40	113.52	1.24

Table 9.4: Timing results (ms) for the concatenation operation (32 NODES)

Message Size	EXPRESS	iPSC/2	Timing Ratio
10	23.10	5.06	4.57
100	53.16	9.69	4.30
200	59.56	15.61	3.09
256	70.80	28.17	2.51
512	98.54	53.78	1.78
1024	158.04	111.14	1.42
2048	278.04	223.20	1.25

4.10 Vector Send/Recv. Operations

Vector read and vector write routines are used to reading and writing vectors with an additional facility of transmitting non-contiguous blocks of data. These routine can be called from host and node processors. At the sending node, ITEMS objects each of size SIZE bytes separated by OFFSET are transmitted. On the receiving end, the received data is read into memory blocks separated by OFFSET. However, the basic functionality of vector read and vector write is same as KXREAD and KXWRIT. iPSC/2 does not provide such primitives. To make a fair comparison, we performed array packing with CSEND and array unpacking with CRECV at the sending and receiving nodes, respectively. The implementation includes the option of packing and unpacking at either ends.

EXPRESS Vector Send/Recv routine :

INTEGER FUNCTION KXVREAD(IBUF, SIZE, OFFSET, ITEMS, SRC, TYPE)

INTEGER FUNCTION KXVWRI(IBUF, SIZE, OFFSET, ITEMS, DEST, TYPE)

Equivalent iPSC/2 routine :

The equivalent iPSC/2 operation written for this purpose uses the following routines along with extra array packing and unpacking.

SUBROUTINE CRECV(TYPE, BUF, LENGTH)

SUBROUTINE CSEND(TYPE, BUF, LENGTH, -1, PID)

Timing results :

Table 10 shows the timing results for send/receive operation between node 0 and node 1 for message size ranging from 10 to 8192. The value of the offset at both the sending and receiving end was selected as 2.

Table 10 : Timing results(ms) for the vector send/receive operation (offset = 2 at source and destination)

	EXPRESS		iPSC/2		Timing Ratio	
Message Size	send	recv.	send	recv.	send	recv.
10	1.370	3.120	0.390	0.720	3.51	4.33
128	3.210	6.370	1.500	2.600	2.14	2.45
256	5.200	9.840	2.440	4.220	2.13	2.33
512	9.810	14.780	4.250	7.430	2.31	1.99
1024	19.130	24.600	7.870	13.910	2.43	1.77
2048	37.610	44.250	15.190	26.680	2.48	1.66
4096	75.340	83.600	29.890	52.340	2.52	1.60
8192	150.730	162.330	59.230	103.760	2.56	1.56

From Table 10, we observe that the performance of EXPRESS primitive for vector exchange is very poor. In spite of the fact that the execution time for iPSC/2 primitive includes the array packing and unpacking time, it is still less than the execution time of EXPRESS primitive.

4.11 Vector Exchange Operation

The vector operation for send/receive between any two nodes, as described above in section 4.10, can be accomplished in a bidirectional manner. This way two nodes can exchange vectors by using a single routine. For EXPRESS, this can be done using EXPRESS primitive described below. For iPSC/2, no such primitive exists and, therefore, CSENDRECV is used for two way data exchange and array packing is done at both ends to make comparison with the equivalent EXPRESS routine.

EXPRESS Exchange routine :

INTEGER FUNCTION KXVCHAN(IBUF, ISRC, ITYPE, OBUF, OLEN, ODEST, OTYPE)

Equivalent iPSC/2 operation :

The equivalent iPSC/2 operation written for this purpose uses the following routine along with extra array packing and unpacking option.

INTEGER FUNCTION CSENDRECV(TYPE, SBUF, SLEN, TONODE, TOPID, TOTYPE, RBUF, RLEN)

Timing results :

Timing results for two way vector exchange operation between node 0 and node 1 for message size ranging from 10 to 512, are given in Table 11.

Table 11: Timing results for the vector exchange operation
(no offset at source and destination)

	EXPRESS		iPSC/2		Timing Ratio	
Message Size	send	recv.	send	recv.	send	recv.
10	4.13	4.15	1.01	0.94	4.09	4.42
64	39.34	39.46	2.35	2.18	16.79	18.10
128	136.79	136.77	3.315	3.025	41.26	45.21
256	509.20	509.23	5.14	4.84	99.07	105.21
512	1958.00	1957.50	8.85	8.53	229.54	229.48

The EXPRESS routine KXVCHAN is extremely slow, as shown in Table 10. The use of this routine is not recommended.

5. The Performance Comparison of Gaussian Elimination by using EXPRESS and iPSC/2

In order to compare the performance of the two programming model with real applications, we wrote and implemented a Gaussian Elimination algorithm with partial pivoting. The algorithm is based on row partitioning of the coefficient matrix. The algorithm was implemented with two programs written under iPSC/2 primitives and EXPRESS primitives. The main difference between two programs lies in the use of communication primitives like broadcast and global reduction operations such as performing the global addition and finding the global maximum. The execution time of both programs versus various matrix sizes using 8 nodes are shown in Figure 5. Figure

6 and Figure 7 show the execution time of Gaussian Elimination for 16 and 32 nodes, respectively.

For 8 processors, the difference in execution times of EXPRESS and iPSC/2 is 1 to 2 seconds. However, EXPRESS gets relatively slower for larger number of processors. The execution time, with larger number of processors is higher for the EXPRESS version. This difference becomes 2 to 3 seconds if 16 processors are used. For 32 processors, the difference becomes up to 10 seconds for a matrix size of 512. The EXPRESS primitives used in this program include KXCOMB, KXREAD and KXWRIT which are substituted for equivalent iPSC/2 primitives, GSSUM, CSEND and CRECV. The timing results indicate that these primitives of EXPRESS and iPSC/2 perform almost identical, with the exception of send operation where CSEND is faster than KXWRIT. However, the increased execution time of EXPRESS version of Gaussian Elimination program is due to the fact that this program uses broadcast communication in such a way that the origin of communication is not known to the receiving processors. This can be easily implemented with iPSC/2 primitive, CSEND and CRECV, by calling these routines only once. The EXPRESS broadcast primitive, KXBROD could not be used in this case because the format of the routine requires the receiving processors to know the id of the sending processor. As a result, KXREAD and KXWRIT have been used. Broadcast was carried out by making a call to these routines for each processor. As a results, the execution time increases if this technique is used for large number of processors.

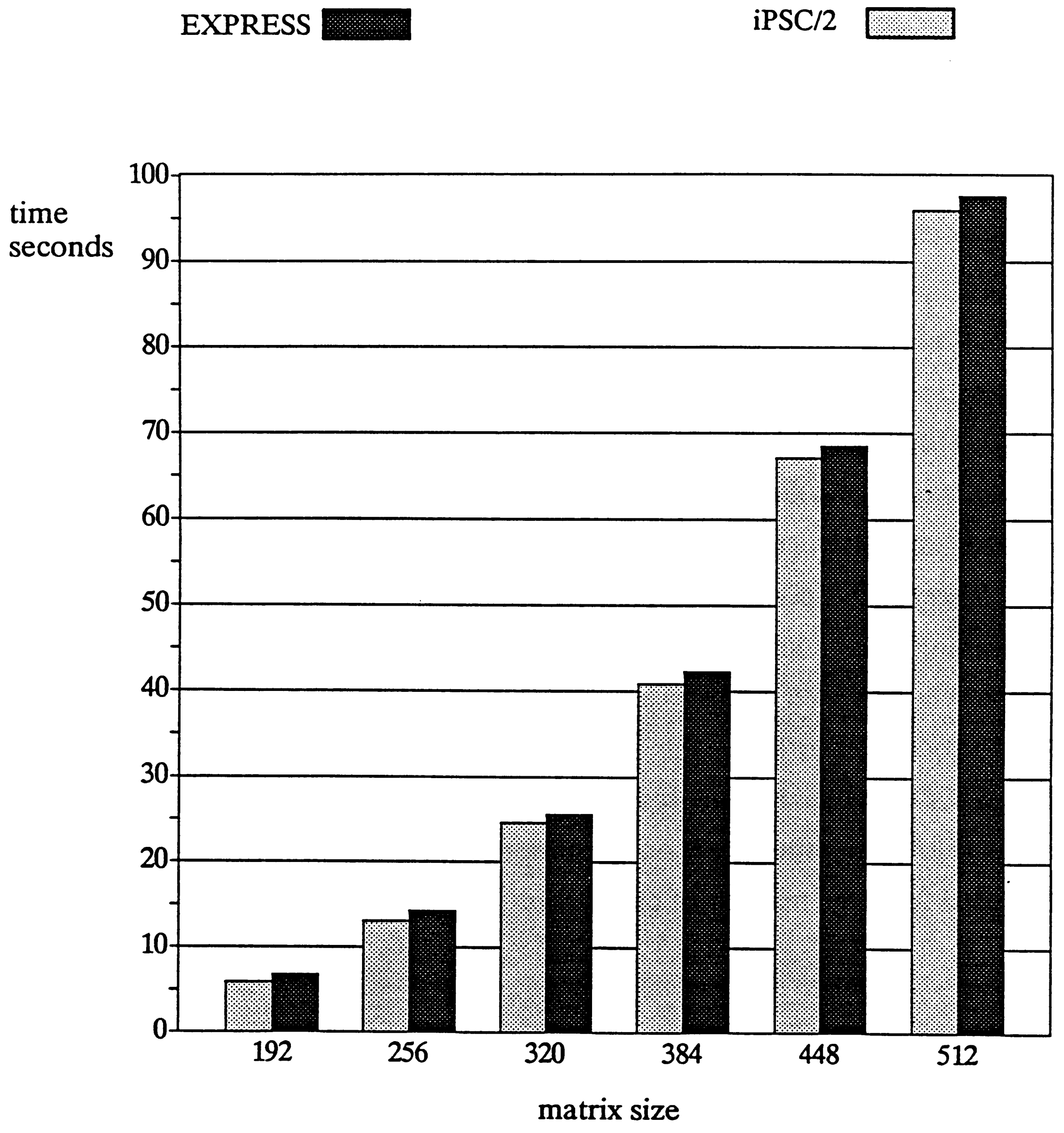

Figure 5: Performance of Gaussian Elimination on 8 Node Hypercube

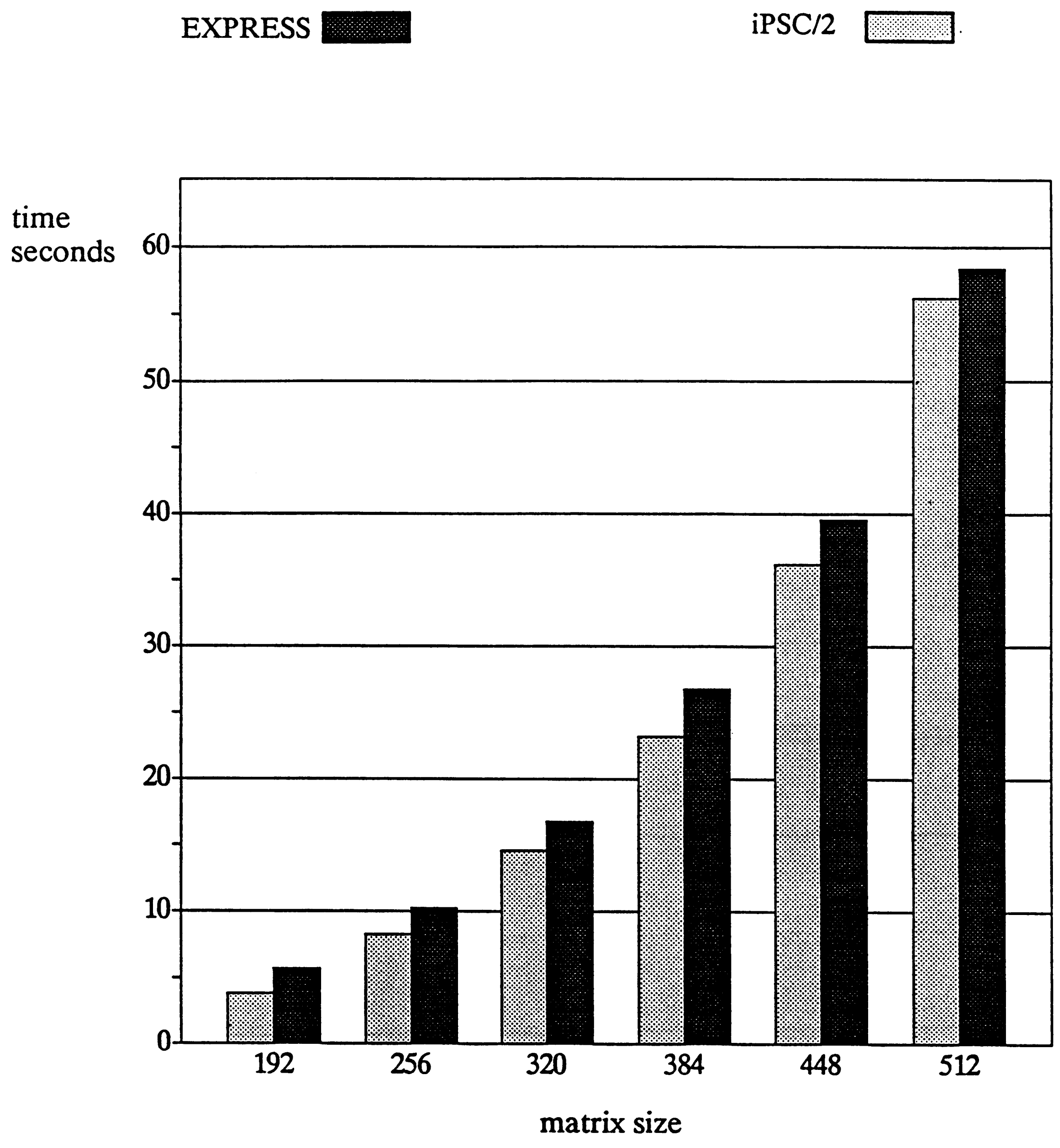

Figure 6: Performance of Gaussian Elimination on 16 Node Hypercube

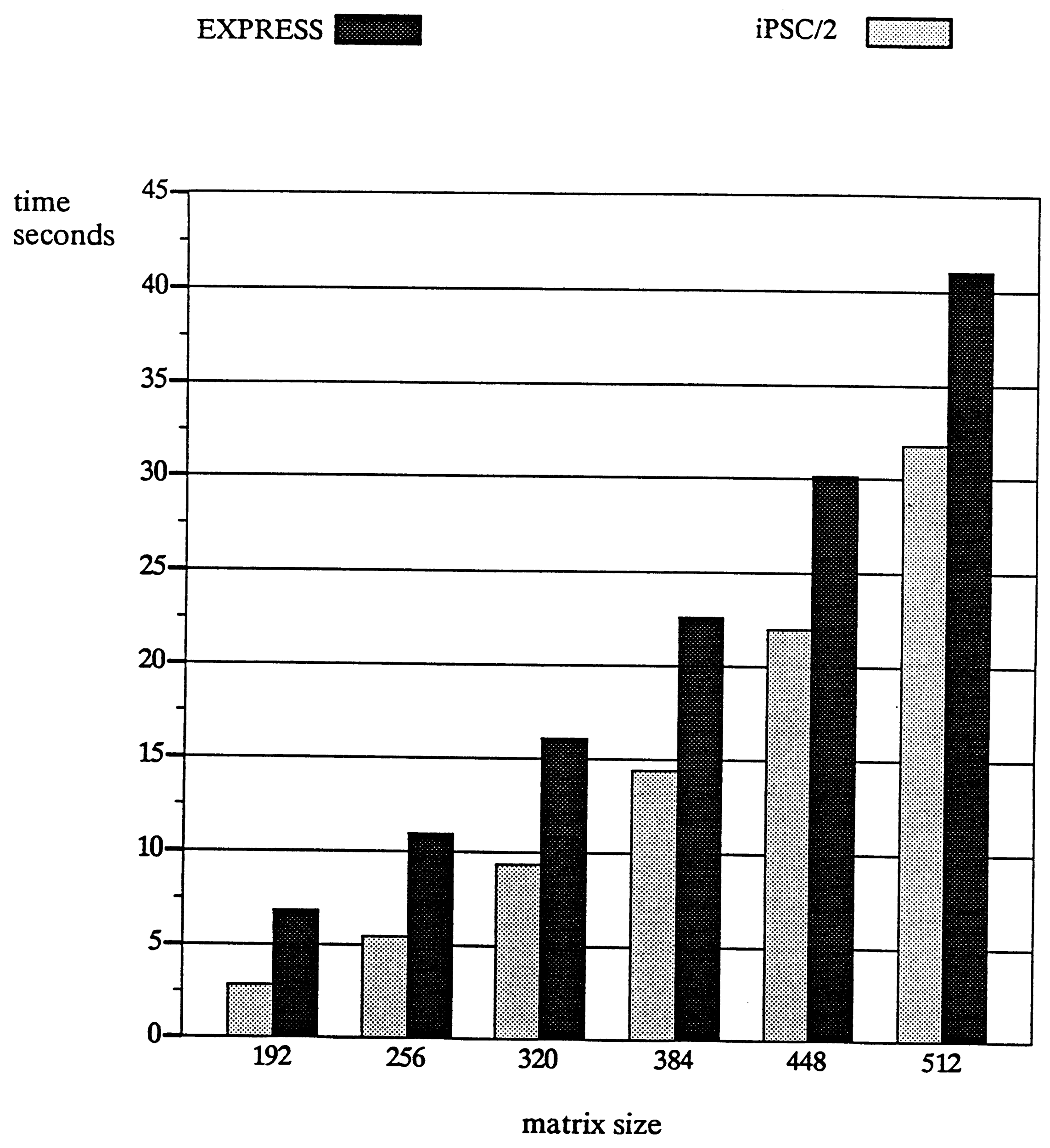

Figure 7: Performance of Gaussian Elimination on 32 Node Hypercube

6. Conclusions

In this report, we have presented a comparison of EXPRESS and iPSC/2 programming models. We have done this comparison by benchmarking various EXPRESS primitives as well as their equivalent iPSC/2 primitives. We have considered various environments, such as host–to–node and node–to–node communication, for testing the performance of these primitives and have analyzed the effect of data size on overall timing. For making a fair comparison, by writing new programs, we also implemented some of the equivalent operations of EXPRESS, which are not provided by iPSC/2. We considered both short and long messages.

Vector functions are useful EXPRESS facilities which are not provided by iPSC/2. On the other hand, EXPRESS has not implemented non–blocking send and receive primitives on the hypercube system so far. The EXPRESS broadcast primitive KXBROD requires the receiving processors to know the id of the sending processor, whereas iPSC/2 primitive CRECV does not. In some cases, such as in the Gaussian Elimination program, the receiving processors do not know the source of broadcasting, and KXBROD cannot be used.

The benchmark comparison of two programming models reveals many insights which can be summarized as follows.

- The iPSC/2 primitives clearly outperform EXPRESS primitives in most of the cases. A few exceptions include the node to node exchange for large messages.
- However, in many cases, the performance of EXPRESS primitives is comparable with iPSC/2 primitives.
- The most important performance measure is the node to node communication time. The comparison shows EXPRESS primitives are 30% to 70% slower than iPSC/2 primitives.
- The receiving time of two models is comparable but the sending time of EXPRESS is higher.

EXPRESS provides a portable environment for parallel programming on different parallel machines with a rich set of utility functions. The cost of using EXPRESS is larger overhead. Larger overhead leads to worse performance, or it requires larger granularity. With the exception of a few cases, the timing ratio of EXPRESS and

iPSC/2 primitives is in the range of 1 to 2.5 (The EXPRESS vector exchange is extremely slow). However, we believe that is an implementation error). Assume communication time is 10% of total execution time, it causes less than 10% performance degradation. Considering its functionality, the overhead of EXPRESS appears not too large and affordable.

Acknowledgements

We thankfully acknowledge the support of Center for Research on Parallel Computation. This work was supported by a grant from National Science Foundation under Cooperative Agreement No. CCR–8809165.

References

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

[2] Express Fortran Reference Guide, Parasoft Corporation, 1990.

[3] iPSC/2 Fortran Language Reference Manual, Intel Corporation, 1989.

[4] I. Angus, G. Fox, J. Kim, D. Walker, *Solving Problems on Concurrent Processors, Volume II*, Prentice Hall, Englewood Cliffs, New Jersey, 1990.

[5] David Bradley, "First and Second Generation Hypercube Performance," Report No. UIUCDCS–R–88–1455, Department of Computer Science, University of Ilinois at Urbana–Champaign, Sept. 1988.

6. Conclusions

In this report, we have presented a comparison of EXPRESS and iPSC/2 programming models. We have done this comparison by benchmarking various EXPRESS primitives as well as their equivalent iPSC/2 primitives. We have considered various environments, such as host-to-node and node-to-node communication, for testing the performance of these primitives and have analyzed the effect of data size on overall timing. For making a fair comparison, by writing new programs, we also implemented some of the equivalent operations of EXPRESS, which are not provided by iPSC/2. We considered both short and long messages.

Vector functions are useful EXPRESS facilities which are not provided by iPSC/2. On the other hand, EXPRESS has not implemented non-blocking send and receive primitives on the hypercube system so far. The EXPRESS broadcast primitive KXBROD requires the receiving processors to know the id of the sending processor, whereas iPSC/2 primitive CRECV does not. In some cases, such as in the Gaussian Elimination program, the receiving processors do not know the source of broadcasting, and KXBROD cannot be used.

The benchmark comparison of two programming models reveals many insights which can be summarized as follows.

- The iPSC/2 primitives clearly outperform EXPRESS primitives in most of the cases. A few exceptions include the node to node exchange for large messages.
- However, in many cases, the performance of EXPRESS primitives is comparable with iPSC/2 primitives.
- The most important performance measure is the node to node communication time. The comparison shows EXPRESS primitives are 30% to 70% slower than iPSC/2 primitives.
- The receiving time of two models is comparable but the sending time of EXPRESS is higher.

EXPRESS provides a portable environment for parallel programming on different parallel machines with a rich set of utility functions. The cost of using EXPRESS is larger overhead. Larger overhead leads to worse performance, or it requires larger granularity. With the exception of a few cases, the timing ratio of EXPRESS and