17th Int'l Symp. on Computer Architecture, May 1990.

Performance Measurement and Trace Driven Simulation of Parallel CAD and Numeric Applications on a Hypercube Multicomputer*

Jiun-Ming Hsu and Prithviraj Banerjee

Center for Reliable and High-Performance Computing
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Urbana, IL 61801

ABSTRACT

This paper presents the performance evaluation, workload characterization and trace driven simulation of a hypercube multicomputer running realistic workloads. Six representative parallel applications were selected as benchmarks. Software monitoring techniques were then used to collect execution traces. Based on the measurement results, we investigated both the computation and communication behavior of these parallel programs, including CPU utilization, computation task granularity, message interarrival distribution, the distribution of waiting times in receiving messages, and message length and destination distributions. The localities in communication were also studied. A trace driven simulation environment was developed to study the behavior of the communication hardware under real workload. Simulation results on DMA and link utilizations are reported.

1. INTRODUCTION

Hypercube multicomputers have recently offered a cost-effective and feasible approach to supercomputing by connecting a large number of low-cost processors with direct links [1]. Each processor has its own local memory. Processes running on these processors communicate via message passing. This type of architecture is more readily scaled up to large numbers of processors than multiprocessor designs based on globally shared memory. Implementations of the hypercube architecture range from experimental prototype systems [2], to commercially available systems from Intel [3], Ametek, and NCUBE.

The evaluation of performance of parallel machines such as hypercubes is extremely important for exploring parallel program characteristics and parallel architecture behavior. One of the goals of our research was to implement a general performance evaluation environment for hypercubes based on software monitoring. In order to do an accurate evaluation, we decided to do some measurements of several realistic parallel applications (four in VLSI CAD and two in numerical domain) written for hypercubes. We used an Intel iPSC/2 hypercube as a testbed for our study.

On the basis of the measurements, we studied issues in both computational and communication behavior of realistic parallel programs on hypercubes. The computational characteristics that we measured were the statistics on CPU utilization, relative system time, user application time, idle time of various CPUs in the parallel system, statistics on task granularity between message generation, etc. The communication characteristics that we measured included the distribution of interarrival times of messages, the distribution of waiting times in receiving messages, and message length and destination distributions.

The motivation behind these measurements was to validate the assumptions typically made by researchers in the parallel computer systems modeling area. The application of queueing network models to computer network performance analysis typically assume some kind of message arrival interval, length, and destination distributions. The most common assumptions are Poisson message arrival, exponentially distributed message lengths, and evenly distributed message destinations [4]. Researchers in the load balancing area make various assumptions about task granularity and intertask dependencies in computational task graphs that are mapped on to processors [5]. We wanted to verify if indeed the above assumptions are valid, and if not, what distributions more accurately model the real world applications on hypercubes.

In experimental studies of hypercube performance [6, 7], simulations have been performed using various synthetic communication benchmarks, which assume different kinds of message interval, length and destination distributions. Again, based on the results of the measurement, we can verify whether these synthetic benchmarks are close to real applications by modeling the empirical data using statistical distributions [8, 9]. The resulting model will also help us in designing more realistic synthetic benchmarks.

The communication patterns of the parallel programs can critically affect the performance of the message passing machines, hence we were also interested in the localities of message destination and length. Models for both the spatial and temporal localities of message destinations have been proposed before in [10]. In this study we propose a model for the locality of message size based on the LRU stack model [11]. The accuracy of this model was verified by measurement results.

In distributed-memory multicomputers like hypercubes, synchronization and data sharing are achieved by explicit message passing. Hence, the speed and efficiency of communication are very important in the overall performance of such machines. In this study, we also investigate the performance of the communication channels in detail. The behavior of the communication hardware components (DMA channels and communication links) is not easily observable via software monitoring. Therefore a

* This research was supported in part by the National Science Foundation Presidential Young Investigator Award under Grant NSF MIP 86-57563 PYI, and in part by an equipment grant NSF CCR 87-05240.

simulation environment was developed to model the hardware and software components of typical hypercube nodes. Experiments were then run using communication traces (message transmission patterns) collected from the earlier part of this work. We were therefore able to obtain reasonably accurate performance information of the communication channels.

The benchmarks used in the measurement are briefly described in Section 2. Section 3 gives an overview of the measurement facilities. Section 4 reports the results of the measurement and the characteristics of the communication workload. In Section 5, we present the trace driven simulation methodology and results.

2. BENCHMARKS

Six hypercube applications are used as benchmarks, which solve real-life VLSI CAD and numerical problems. Different data partitioning schemes are employed in these applications, and the communication patterns vary from ring, nearest neighbors, distance-two, to broadcast communication. The communication workload ranges from very light to very heavy. Together, these applications represent a typical workload of hypercube multicomputers. Four parallel VLSI CAD programs are used, which are: a standard cell placement program, a channel routing program, a VLSI circuit extractor, and a test pattern generator. In addition, two numeric applications, QR factorization and Fast Fourier Transformation, are also used.

The parallel standard cell placement program [12] is based on simulated annealing technique whose objective is to minimize the total estimated wirelength in a standard cell layout. Processors pair up to perform parallel moves, and synchronize with each other using a ring-based broadcast mechanism. The parallel channel routing program [13] is also based on simulated annealing and uses similar communication patterns to minimize the number of horizontal tracks in a channel. The parallel test pattern generator [14] is based on a parallel branch and bound algorithm. The scheduling and control of the parallel search is done by one processor, while other nodes do the searches in parallel. The parallel circuit extraction program [15] consists of two phases. The first phase, dominated by communication, is the distribution of data from the host to the node processors in a tree fashion. The second phase is the circuit extraction phase which is computationally intensive.

The parallel QR factorization algorithm [16] maps the matrix onto a ring of processors and eliminates matrix elements in parallel. The parallel Fast Fourier Transform (FFT) program [17] involves an even distribution of the input points among the processor nodes initially. The nodes subsequently perform the iterative sequential FFT algorithm on the points in parallel. At the end of each step, messages are exchanged across nodes to set up the points for the next iteration.

In each of the above applications, with the exception of the circuit extractor, only the main execution part is measured. The downloading of the data and the uploading of the results are not included. For the extractor, both the downloading of the data and the execution part are traced.

3. MEASUREMENT FACILITIES

The system on which we performed the measurement is a 16 node Intel iPSC/2 hypercube [3], where the application programs access the message passing capabilities through two sets of system calls: 1) blocking message calls, *csend()* and *crecv()*, and 2) non-blocking message calls, *isend()* and *irecv()*. Messages of 100 bytes or less (short messages) are unsolicited. Long messages must request the receiving buffer from the destination node first before sending the message, so they follow a three-trip protocol — request, acknowledge, send.

We have implemented a software monitoring tool in the iPSC/2 processor nodes. The monitoring facility can be used to tune the software for better performance, to investigate resource utilization, and to determine the characteristics of computation and communication workload. The measurement methodology basically followed the same approach as [18], which is software monitoring in the operating system level. The user program need not be modified except that two commands, *monitor_init()* and *monitor_end()*, are added to the node program to initiate and terminate the software monitoring facilities.

The instrumentation is done by inserting *probe points* at various places in the node operating system. An event is triggered during the execution of the application program when any of the probe points is hit. The monitoring facilities record each event onto a trace buffer in the memory together with the time-stamp and an optional number of parameters of the particular event. At the end of the monitoring, when the *monitor_end()* command is executed, the trace buffer is sent to a background process in the host which stores the trace in the file system. One trace file is generate by each of the processor nodes.

The places where we insert probe points are:

(1) entry and exit points of system calls (message and non-message),

(2) entry and exit points of interrupt service routines (message and non-message),

(3) points where the CPU enters and leaves idle state,

(4) points where the operating system performs message related activities, like start sending a message, set up receiving for a message, and

(5) entry and exit points of buffer operations, e.g. buffer allocation and buffer copying.

The first three types of probe points enable us to observe the CPU state and the specific operating system routine executed. The fourth one provides the information about communication activities, and the last type of probe points help us determine the cost of various buffer operations.

Software monitoring slows down the execution of the benchmarks by 0.2 to 35 percent, as shown in Table 1. It also shows the input data used in the measurement, average number of messages sent and received, average message size, size of trace file, and number of trace events, of each node. Since the emphasis of the monitor is on communication and system activities, the percentage of the overhead is directly proportional to the rate of message and system calls. The channel routing program has a high message transmission rate and the test pattern generation program makes frequent system calls, therefore their overhead percentages are very high.

4. PERFORMANCE MEASUREMENT RESULTS

This section presents the results of the measurement of the six benchmarks running on an eight node hypercube. We show the utilization of the CPU and the characteristics of the computation and communication workload. Models for the locality of message length and the distributions of various time intervals are also proposed.

4.1. CPU Utilization

The utilization of the CPU gives a good indication of the efficiency of the parallel algorithm. Table 2 shows the CPU utilization of the six applications running on a eight node hypercube. The total execution time of the CPU is divided into three parts, which are system, user and idle times. For each of the three parts, the average percentage is shown together with the maximum and minimum among the eight nodes.

The system time includes both the message-related and message-unrelated calls. In the benchmarks, most of the system calls are message-related, hence the system percentage is proportional to the rate of message sending and receiving. The variations in the user time percentages indicate the computation workload distribution of the applications. The CPU idle time occurs mostly in waiting for the completion of message sending and receiving. Lower idle time translates to more efficient use of system resources.

The channel router is a communication intensive program. In the test generation program, the communication and computation workloads are not balanced. This is because processor zero acts as the scheduler of the parallel branch and bound search, hence it has the lowest user time, highest system and idle time percentages. The circuit extraction program, which does explicit load balancing, and the numeric applications, QR and FFT, are well load-balanced. The first phase of the circuit extractor is I/O intensive, and the second phase is computation intensive. Due to its large data size, the FFT program is very computation intensive.

Figure 1 shows the CPU utilizations of the cell placement and QR programs when the number of processors increases from 4 to 16. As expected, for both of the programs, the user time percentage goes down and the system time percentage goes up when the size of the hypercube grows. This is because the granularity of the computation task becomes smaller. Note that the user time percentage of the placement program declines faster than that of the QR program because the placement program sends more messages per node when the hypercube size grows. Figure 2 shows the CPU utilizations of the QR program when the data size increases. It again confirms that when the task granularity becomes larger, the user CPU utilization also increases.

4.2. Message Length and Destination

Figures 3 and 4 summarizes the message destination and length distributions of node zero for four of the applications: placement, extraction, QR and FFT. The cell placement program communicates only with distance one and two nodes, with most of the messages going to one distance-one node. The message length varies from 0 to 4K bytes, but most are under 100 bytes. The circuit extractor sends almost all the messages to distance-one nodes, mostly with length less than 100 bytes or around 16K bytes. The nodes in the QR factorization program communicate in a ring fashion, sending 1028 byte messages to their successors and 1024 byte messages to their predecessors. In the FFT program, the nodes send equal amount of messages to all of the distance-one nodes, all with the same length.

Table 3 summarizes the spatial locality of message passing for the six benchmarks on an 8 node hypercube. Except for the test generation program, all the benchmarks send most of the messages to the nearest neighbors. The communication patterns do not change for a 16 node configuration. This indicates that the message destination has spatial locality which favors nearest neighbors. Evenly distributed message destination is often assumed in both experimental [7] and analytical [4] approaches of computer network performance analysis. This assumption is not true in most of the applications for hypercubes because the programmers often try to avoid sending long distance and global messages.

In addition to the spatial locality, we were also interested in the temporal locality of the message destination. We model the temporal locality using the *least recently used* (LRU) stack model [10, 11], in which the message destination is treated like memory address. In the LRU stack model of stack size n, each node has a stack which contains the n most recent message destinations. If the destination of the next message is in the stack, it is called an LRU stack hit, otherwise it counts as a miss. In Table 4, we show the average LRU stack hit rate of the six applications when stack size varies from one to four. Four out of the six benchmarks have a high hit rate when the stack size is only one. All programs have a hit rate of 85 percent or more when the stack size is three. In the FFT program, each node alternately sends messages to the three distance-one neighbors and the destinations always hit the third stack location.

If the message length is also treated like a memory address, then the sizes of messages also have spatial and temporal localities. From the distributions of the message length in Figure 4, it is clear that the lengths concentrate around only one or two peaks in every application, which shows that the lengths have high spatial locality. To investigate the temporal locality of the message length we propose a model called the **Most Recent Message Length** (MRML) stack model, which is based on the LRU stack model. In the MRML stack model of stack size m, each node has several stacks each corresponds to one possible message destination and contains the m most recent message sizes send to that node, with the most recent length in the first location. The message is considered a MRML

Application Program	Message Distance (%) 1 Hop	2 Hop	3 Hop
Placement	88.5	11.5	0
Routing	91.9	8.1	0
Test Gen.	52.8	37.2	10.0
Extraction	99.4	0.6	0
QR	100	0	0
FFT	100	0	0

Table 3. Average Message Distance Distribution.

Application Program	Destination LRU Stack Hit Rate (%) DSS=1	DSS=2	DSS=3	DSS=4
Placement	83.52	85.52	85.96	86.35
Routing	81.82	90.90	99.96	99.96
Test Gen.	87.68	88.75	90.15	91.27
Extraction	39.51	74.05	98.51	98.78
QR	87.94	99.90	99.90	99.90
FFT	0	0	99.00	99.00

Table 4. Message Destination Hit Rates as a Function of Destination LRU Stack Size.

Application	MRML Stack Hit Rate (%)				MRML Stack Hit Rate (%) Min. Msg. Size = 100 Bytes			
Program	LSS=1	LSS=2	LSS=3	LSS=4	LSS=1	LSS=2	LSS=3	LSS=4
Placement	82.0	87.9	90.6	91.7	89.3	92.2	94.1	94.8
Routing	91.2	94.1	94.6	95.0	99.9	99.9	99.9	99.9
Test Gen.	78.1	86.1	93.7	96.5	88.4	92.2	97.3	98.3
Extraction	48.1	74.1	82.3	85.8	49.1	74.1	82.3	85.8
QR	99.9	99.9	99.9	99.9	99.9	99.9	99.9	99.9
FFT	99.0	99.0	99.0	99.0	99.0	99.0	99.0	99.0

Table 5. Message Length Hit Rates as a Function of MRML Stack Size.

Application	Destination and Length Stack Hit Rate (%) Min. Msg. Size = 100 Bytes					
	MRML Stack Size = 1			MRML Stack Size = 2		
Program	DSS=1	DSS=2	DSS=3	DSS=1	DSS=2	DSS=3
Placement	80.93	82.59	82.77	81.62	83.37	83.62
Routing	81.78	90.85	99.90	81.78	90.85	99.90
Test Gen.	76.50	77.57	78.98	80.26	81.33	82.74
Extraction	15.91	33.00	48.96	28.28	54.11	73.95
QR	87.94	99.90	99.90	87.94	99.90	99.90
FFT	0	0	99.00	0	0	99.00

Table 6. Message Destination and Length Hit Rate.

stack hit if its length is smaller than or equal to the longest of the m last messages send to the same destination, i.e. the largest in the MRML stack. Note that this model is for the upper bound of the message length.

For example, if a node x sends messages to four other nodes, there will be four MRML stacks for node x each corresponds to one destination. Stack $x.y$ contains the sizes of the most recent 3 messages node x sent to y, assuming a stack size of 3. If the current content of $x.y$ is (6,10,5) and the next message is of size 7, then it is a MRML stack hit. The stack distance is two, since 7 is smaller than the second stack location but larger than the first one. The new $x.y$ stack will be (7,6,10). If the next message length is 12, then it counts as a miss.

Table 5 shows the average MRML stack hit rate of the six benchmarks when the stack size increases from one to four. The left hand side of this table reports the pure MRML stack hit rate without any limitation on minimum message length. The hit rates of QR and FFT are almost 100 percent, since, for each destination node, the message length is constant. The placement and routing programs also have high hit rates when the stack size is only one.

In the right hand side of Table 5, following the message transmission protocol of iPSC/2, we assume that the minimum buffer size is 100 bytes, hence messages shorter than 100 bytes will be treated as 100 bytes. The hit rates of placement, routing and test generation programs improves substantially. The circuit extraction program often sends a short message and then a very long message alternately to each destination node, therefore the hit rate is very low when the stack size is one, and the minimum message size requirement does not make any difference.

We now show the predictability of the communication patterns by combining both of the models for the temporal localities of message destination and length. The message is considered a hit only when its destination is in the LRU stack *and* its length hits the MRML stack. Table 6 shows the average message hit rates of the benchmarks when the MRML stack size increases from one to two, and the destination LRU stack size varies from one to three. We also assume the minimum message size to be 100 bytes. Compared to the pure destination LRU stack hit rates shown in Table 4, the hit rates for the placement and routing programs decline a little bit, while those of the numeric applications stay the same since the length prediction always hits. For the test generator and the circuit extractor, the hit rates drop by a large amount, but recover substantially when the MRML stack size becomes two.

These measurements indicate that message length and destination have high spatial and temporal localities, hence the communication patterns are highly predictable. By utilizing these locality properties, the cost of message communication could be reduced by intelligent routing

Number of Stages	Model		
	Normal	Hypoexp.	Gamma
1	63.1	46.6	50.1
2	6.5	11.1	12.6
3	1.7	7.5	4.1

Table 7. SSE of Curve Fitting on Message Interval Dist. of Node 0 of the Cell Placement Program.

Number of Stages	Model		
	Normal	Hypoexp.	Gamma
1	5.3	12.1	6.0
2	1.3	7.5	2.7

Table 8. SSE of Curve Fitting on Message Interval Dist. of Node 0 of the QR Factorization Program.

Appl. Prog.	Distributions	Two-stage Normal pdf Model Parameters					
		a_1	μ_1	σ_1	a_2	μ_2	σ_2
Placement	Msg. Interval	0.49	0.57	0.07	0.51	1.45	1.01
	Computing Time	0.52	0.07	0.02	0.48	0.14	0.07
	Waiting Time	0.54	0.18	0.02	0.46	0.47	0.42
QR	Msg. Interval	0.84	5.77	0.60	0.16	6.23	0.16
	Computing Time	0.71	3.38	0.50	0.29	3.98	0.19
	Waiting Time	1.00	1.35	0.05	-	-	-

Table 9. Two-stage Normal pdf Model Parameters for Placement and QR (Node 0).

controllers and efficient buffer management strategies. In a circuit-switched message routing paradigm, the circuit can be kept connected even after the message is transmitted. If the next message also goes to the same destination, this circuit can again be used, thereby reducing circuit set-up time. Software overhead can also be reduced by predicting the sizes of the incoming messages and pre-allocating buffers for them.

4.3. Communication and Computation Workload

In experimental and analytical studies of computer network performance, different kinds of message interarrival time distributions are often assumed, including exponential [4], normal distribution [7], and constant interval [6]. We wanted to verify these assumptions by analyzing the data collected from the measurements.

Figure 5 shows the distribution of the time interval between message sending of node zero for the six benchmarks. The interarrival time between messages $i-1$ and i, t_i, can be broken down into three parts, which are the computation time, c_i, the waiting time for receiving messages, w_i, and the system time for processing messages, s_i. Hence, $t_i = c_i + w_i + s_i$. Figures 6 and 7 show the distributions of the computation time and waiting time interval of node zero. The system time for processing messages can be approximated by a linear function of the message size, $s_i = f(l)$, hence it is very predictable and will not be discussed here.

The message interarrival distribution of the placement program has two peaks, which correspond to the two types of messages it sends, ring broadcast and cell exchange. The distributions of the placement program are similar to those of the channel router, because they use similar algorithms and have similar message transmission patterns. The circuit extractor sends infrequent messages and, in the first phase, spends most of the time waiting for messages from the host. Its computing time interval is irregular. As expected, the distributions of the numeric algorithms, especially those of the FFT, are very regular. Each of the three distributions of the QR program has a single peak, and those of the FFT have two peaks.

Due to the multiple peaks, the distributions clearly can not be modeled by simple exponential, normal or gamma probability density functions (pdf). Therefore, we model the distributions by multi-stage pdf's [8], defined as:

$$f(t) = \sum_{i=1}^{n} a_i \; g_i(t),$$

where $a_i \le 1$ is the weight for each stage, $\sum_{i=1}^{n} a_i = 1$, and n is the number of stages. The probability density functions $g_i(t)$ can all be chosen from one of the following:

(1) Normal:

$$g_i(t) = \frac{1}{\sigma_i \sqrt{2\pi}} \exp\left[-\frac{1}{2}\left[\frac{t-\mu_i}{\sigma_i}\right]^2\right], \quad t \ge 0,$$

where $\mu_i > 0$, and $\sigma_i > 0$.

(2) Shifted hypoexponential:

$$g_i(t) = \begin{cases} 0, & t \le s_i, \\ \dfrac{\lambda_{1i}\lambda_{2i}}{\lambda_{1i}-\lambda_{2i}} \left[e^{-\lambda_{1i}(t-s_i)} - e^{-\lambda_{2i}(t-s_i)}\right], & t > s_i, \end{cases}$$

where $\lambda_{1i} \ne \lambda_{2i}$, and $s_i > 0$ is the offset in time.

(3) Shifted gamma:

$$g_i(t) = \begin{cases} 0, & t \le s_i, \\ \dfrac{\lambda_i^{\alpha_i} \; (t-s_i)^{\alpha_i - 1} \; e^{-\lambda_i (t-s_i)}}{\Gamma(\alpha_i)}, & t > s_i, \end{cases}$$

where $\alpha_i > 0$, $\lambda_i > 0$ and $s_i > 0$. The gamma function is defined by the integral:

$$\Gamma(\alpha_i) = \int_0^{\infty} x^{\alpha_i - 1} e^{-x} \, dx, \quad \alpha_i > 0.$$

Thus, we have multi-stage normal, multi-stage shifted hypoexponential, and multi-stage shifted gamma pdf's. To determine the best model for the empirical distributions shown in Figures 5 through 7, we applied curve fitting technique (nonlinear regression) [19] to fit the distributions into the above models. Similar studies have been reported by [9] for file usage analysis and [8] for resource utilization. The criterion of the nonlinear regression procedure is

Node 0:
Start node 0.
Compute 350 μSec.
Csend(type 1, length 100, to node 1).
Compute 400 μSec.
Crecv(type 2, length 500).
Compute 450 μSec.
End.

Node 1:
Start node 1.
Compute 500 μSec.
Crecv(type 1, length 100).
Compute 500 μSec.
Csend(type 2, length 500, to node 0).
Compute 400 μSec.
End.

Figure 8. HSIM Input for a Two Node Hypercube.

to minimize the sum of square of errors (SSE) when estimating the parameters of a nonlinear model.

Table 7 shows the SSE of curve fitting for the message interval distribution of the placement program (shown in Figure 5.a). The models used are the three types of distributions with 1, 2 and 3 stages. The results indicate that two or three stage normal pdf models this distribution better than the other two types of pdf's. The same experiment repeated for the message interarrival time distribution of QR factorization program, shown in Table 8, also confirms this observation. By performing nonlinear regression on all the empirical distributions including the computation and waiting time intervals, we found that, while not the best model in every case, the multi-stage normal pdf has better overall performance than the other two models. A two-stage normal pdf models the distributions reasonably well, as can be seen from Tables 7 and 8, therefore it was chosen to model the empirical data.

Table 9 shows the two-stage normal pdf parameters for the distributions of cell placement and QR factorization programs. For the placement program, the two model parameters, μ_1 and μ_2, correspond to the two peaks in its message interarrival distribution, as shown in Figure 5.a. The parameters of its computing time and waiting time distributions indicate that the task granularity is small and the waiting times for messages are also not long. The waiting time distribution of the QR program can be modeled by one normal pdf with a small variance. Since $t_i = c_i + w_i + s_i$, as previously stated, and the waiting time w_i and system time s_i are almost constant, the message interval and computing time distributions are very similar except that one is time shifted by about 1.4 mSec from the other, as can be seen from μ_1 and μ_2 of the two distributions.

In this subsection, we have shown that the distributions of message arrival, computation time and waiting time can be modeled reasonably well by the two-stage normal pdf models. The results of such a study can be useful in the generation of realistic synthetic benchmarks and accurate analytical models for the evaluation of hypercube applications and architectures.

We also note that the statistics on the computation time between messages relate to the computation task granularity, and the statistics on message length and destination distributions represent the inter-task dependencies in computation task graphs. Our results will therefore provide realistic models of task graphs for load balancing researchers.

5. TRACE DRIVEN SIMULATION

5.1. Simulation Methodology

A trace driven simulator for hypercube — HSIM has been developed to study the behavior of the communication hardware under real workload. HSIM, written in C++, is based on CSIM [20], which is a process oriented simulation language. The emphasis of HSIM is in communication activities, therefore the message transmission is modeled in detail. The computation time of CPU between message sending and receiving is derived directly from the traces.

The inputs to HSIM are the *communication traces* which are derived from the execution traces of real applications by a post-processing program. Figure 8 shows a sample input to HSIM for a two node hypercube. Here the two processors exchange two messages while performing some computation in between. Since HSIM actually executes *csend()* and *crecv()* commands, it has to wait for the completion of message transmissions before continuing execution. Hence, it is very realistic in modeling the execution behavior of parallel programs.

HSIM models the architecture and communication protocol of the iPSC/2 [3], which uses circuit-switched message routing scheme. Each communication link consists of two bi-directional, bit-serial channels. Two DMA channels are used to transmit data, one for memory to routing logic transfers (the *outgoing DMA*), the other for transfers in the opposite direction (the *incoming DMA*).

It has been found that the errors are within five percent when comparing the simulation and measurement results on the total execution time and CPU utilization of the application programs. This enables us to justifiably put reliance on the results of the trace driven studies of link and DMA channel behavior.

5.2. Simulation Results

This subsection presents the results of the trace driven simulation on DMA and link utilizations. The inputs to the simulator were converted from the same set of traces that we characterized in Section 4. Table 10 shows the utilizations of the incoming and outgoing DMA channels, each with the minimum, maximum and average utilizations among the eight nodes, and the average service time for each message. Also shown are the overlap between the two channels, the overall DMA utilization and the CPU and DMA overlap percentage. Table 11 shows the link utilization and average bandwidth for each of the

Application Program	Incoming DMA Channel				Outgoing DMA Channel				In/Out	DMA	Average CPU & DMA Overlap
	Utilization (%)			Service Time	Utilization (%)			Service Time	Overlap	Util.	
	Min.	Max.	Avg.	Avg. (μSec)	Min.	Max.	Avg.	Avg. (μSec)	(%)	(%)	(%)
Placement	1.24	1.27	1.25	135.7	1.30	1.34	1.32	143.3	0.15	2.42	0.304
Routing	8.39	8.43	8.41	105.3	8.54	8.63	8.60	107.7	2.11	14.90	1.607
Test Gen.	0.30	3.18	0.72	115.7	0.55	2.61	0.87	172.2	0.04	1.55	0.101
Extraction	1.50	2.74	2.43	1344.6	1.37	4.50	2.48	1331.1	0.01	4.90	1.127
QR	8.19	9.01	8.60	179.3	8.28	9.10	8.67	180.7	0.02	17.26	0.210
FFT	1.71	1.71	1.71	544.6	1.80	1.82	1.81	576.8	0.29	3.22	0.253

Table 10. DMA Utilization (8 Node Hypercube).

Application Program	Node 0						Node 4						Overall	
	Link 0		Link 1		Link 2		Link 0		Link 1		Link 2			
	Util.	BW	Util.	BW	Util.	BW	Util.	BW	Util.	BW	Util.	BW	Util.	BW
Placement	0.38	6.79	0.23	4.71	1.02	15.09	1.02	15.11	0.22	4.33	0.39	7.22	0.55	9.05
Routing	0.98	11.62	0.32	3.80	7.84	94.68	8.18	98.40	0.32	3.83	0.65	7.71	3.06	36.64
Test Gen.	0.55	7.17	1.27	16.31	2.24	28.19	0.60	7.03	1.22	13.45	0.36	4.08	0.48	5.60
Extraction	1.31	35.69	0.20	4.37	0.02	0.24	1.27	35.00	0.26	5.43	1.01	27.68	0.83	22.20
QR	4.57	110.36	0	0	4.09	99.14	4.14	100.79	0	0	4.08	98.83	2.87	69.84
FFT	0.60	15.27	0.60	15.27	0.60	15.27	0.60	15.27	0.60	15.27	0.60	15.27	0.60	15.27

(Utilization in percentage, bandwidth in KBytes/Sec.)

Table 11. Link Utilization (8 Node Hypercube).

Data Size (Cells)	Number of Nodes	CPU			DMA		Link	
		Sys. (%)	User (%)	Idle (%)	In (%)	Out (%)	Util. (%)	BW (KB/Sec)
286	4	2.25	68.99	28.76	0.85	0.90	0.54	9.72
286	8	4.10	62.38	33.52	1.25	1.32	0.55	9.05
286	16	5.10	46.02	48.88	1.50	1.59	0.47	7.18
183	8	10.39	48.31	41.29	2.96	3.18	1.28	17.10

Table 12. Placement Program on Different Number of Nodes and Data Size.

three outgoing links of node zero and four, and the average utilization and bandwidth of all the links.

The DMA utilization table indicates that there is not much overlap between CPU and DMA. Part of the reason is that the DMA utilizations are not high anyway. In addition, most of these benchmarks use blocking communication calls, therefore almost all of the overlap occurs between CPU and the incoming DMA channel. The execution time could be reduced by overlapping communication with computation using non-blocking communication calls.

The channel routing program has high DMA and link utilization, this confirms the measurement result that it is a communication intensive application. Also, both of its CPU/DMA and incoming/outgoing channel overlaps are high. This is not due to explicit non-blocking message calls, but rather, it is caused by the asynchronous communication pattern of the processors where messages arrive when they are doing computation or sending out messages.

QR factorization program is also a communication intensive application, as can be seen from the high DMA and link utilizations. However, in contrast to the channel router, QR has low overlap between CPU/DMA and incoming/outgoing channels. This is because QR has a more regular communication pattern and its computation load is more balanced. It is interesting to note that the channel router has a higher link utilization rate than QR, but its average link bandwidth is almost half of that of QR. The reason is that most of the messages of the channel router are very short, while QR sends long messages of 1K bytes. Hence the data transmission rate of QR is much higher.

The circuit extraction and FFT programs have longer average DMA service times because their average message sizes are larger than the other programs. Except the FFT program, where every node communicates with all of its distance-one neighbors, there exist high variations among the utilizations of the links of each node.

Table 12 gives the resource utilization summary of the cell placement program running on different hypercube sizes and with different input data. The CPU utilization part of the table again confirms the results of measurement, which were shown in Figure 1. As the cube size grows, the program becomes less computation intensive and more communication intensive. Therefore, the user time percentage goes down and the utilization of DMA increases. However, the average utilization and bandwidth decrease despite the fact that there are more messages. This is because the total number of links increases in a higher rate than that of node number. When data size decreases, the program also becomes less computation intensive.

For all the benchmarks except channel router and QR, the DMA and link utilizations are under 5 and 1 percent respectively, and the average bandwidths are well under the 2.8 Megabytes per second maximum. The low utilization rate indicates that a mesh architecture could probably do as well as the hypercube for these programs. In addition, in the iPSC/2 system, the delays in messages are not due to congestion in the links. With the existing hardware, better performance could be achieved just by reducing software overhead. The low transmission rate of small messages also shows that the system is not suitable for exploring low level parallelism in the applications.

6. CONCLUSION

We have investigated the computation and communication behavior of six parallel programs on a message passing multicomputer. An LRU stack model was used to study the temporal locality of message destinations. We also proposed a model for the temporal locality of message length which is called the *Most Recent Message Length* (MRML) stack model. By using these models, we have demonstrated that both the message destination and length have high temporal and spatial localities. These measurements and models have shown that the communication cost can be reduced by intelligent routing controllers and efficient buffer management strategies.

Multi-stage probability density functions were used to model computation task granularity distribution, message interarrival distribution, and the distribution of waiting times in receiving messages. It has been found, by nonlinear regression technique, that two-stage normal pdf's are most suitable in modeling the above distributions.

We also described a trace driven simulator which was used to study the communication activities in detail. The results of simulation show that the utilizations of DMA and links are very low, which indicates that there are still rooms for improvement even with the existing hardware.

By modeling the computation and communication workloads of realistic parallel programs, this study provided us some very important information that will be useful in both analytical and experimental study of hypercube communication networks. Based on the models for distributions and localities, representative synthetic benchmarks and accurate analytical models can be designed which will hopefully produce more precise results. System designers can also benefit from these information in developing more efficient communication hardware and software.

REFERENCES

[1] C. L. Seitz, ''The Cosmic Cube,'' *Communications of the ACM*, vol. 28, pp. 22-33, Jan. 1985.

[2] J. Tuazon, J. Peterson, and M. Pniel, ''Mark IIIfp Hypercube Concurrent Processor Architecture,'' in *Proc. 3rd Conf. on Hypercube Concurrent Computers and Applications*, Pasadena, CA, pp. 71-80, Jan. 1988.

[3] P. Close, ''The iPSC/2 Node Architecture,'' in *Proc. 3rd Conf. on Hypercube Concurrent Computers and Applications*, Pasadena, CA, pp. 43-50, Jan. 1988.

[4] P. Kermani and L. Kleinrock, ''Virtual Cut-Through: A New Computer Communication Switching Technique,'' *Computer Networks*, vol. 3, pp. 267-286, Sep. 1979.

[5] S. H. Bokhari, ''Partitioning Problems in Parallel, Pipelined and Distributed Computing,'' *IEEE Trans. on Computers*, vol. 37, No. 1, pp. 48-57, Jan. 1988.

[6] D. C. Grunwald and D. A. Reed, ''Networks for Parallel Processors: Measurements and Prognostications,'' in *Proc. 3rd Conf. on Hypercube Concurrent Computers and Applications*, Pasadena, CA, pp. 610-619, Jan. 1988.

[7] G. Buzzard and T. Mudge, ''High Performance Hypercube Communication,'' in *Proc. 3rd Conf. on Hypercube Concurrent Computers and Applications*, Pasadena, CA, pp. 600-609, Jan. 1988.

[8] M. C. Hsueh, R. K. Iyer, and K. S. Trivedi, ''Performability Modeling Based on Real Data: A Case Study,'' *IEEE Trans. on Computers*, vol. 37, No. 4, pp. 478-484, April 1988.

[9] M. V.-S. Devarakonda, ''File Usage Analysis and Resource Usage Prediction: a Measurement-Based Study,'' Technical Report CSG-79, Coordinated Science Lab., Univ. of Illinois, Urbana, IL, Dec. 1987.

[10] D. A. Reed and R. M. Fujimoto, in *Multicomputer Networks: Message-Based Parallel Processing*. Cambridge, MA: MIT Press, 1987.

[11] R. L. Mattson, J. Gecsei, D. R. Slutz, and I. L. Traiger, ''Evaluation Techniques for Storage Hierarchies,'' *IBM Systems Journal*, vol. 9, 1970.

[12] J. Sargent and P. Banerjee, ''A Parallel Row-Based Algorithm for Standard Cell Placement with Integrated Error Control,'' in *Proc. 26th Design Automation Conf.*, Las Vegas, NV, Jun. 1989.

[13] R. Brouwer and P. Banerjee, ''A Parallel Simulated Annealing Algorithm for Channel Routing on a Hypercube Multiprocessor,'' in *Proc. Int. Conf. on Computer Design*, Rye Brook, NY, pp. 4-7, Oct. 1988.

[14] S. Patil and P. Banerjee, ''A Parallel Branch and Bound Approach to Test Generation,'' in *Proc. 26th Design Automation Conf.*, Las Vegas, NV, Jun. 1989.

[15] K. P. Belkhale and P. Banerjee, ''PACE2: An Improved Parallel VLSI Extractor with Parametric Extraction,'' in *Proc. Int. Conf. Computer-Aided Design*, Santa Clara, CA, Nov. 1989.

[16] A. Pothen, S. Jha, and U. Vemulapati, ''Orthogonal Factorization on a Distributed Memory Multiprocessor,'' *Proc. 2nd SIAM Conf. on Hypercube Computers and Applications*, pp. 587-596, 1987.

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

[18] D. C. Rudolph, ''Performance Instrumentation for the Intel IPSC/2,'' Technical Report UIUCDCS-R-89-1524, Dept. of Computer Science, Univ. of Illinois, Urbana, IL, July, 1989.

[19] SAS Institute Inc., ''SAS User's Guide: Statistics,'' Version 5, Cary, NC, 1985.

[20] H. Schwetman, *CSIM Reference Manual (Revision 13)*. Austin, TX: Microelectronics and Computer Tech. Corp., May, 1988.