Proceedings of the 1988 Winter Simulation Conference
M. Abrams, P. Haigh, and J. Comfort (eds.)

A simulation of a store-and-forward distributed network of transputers

Janice R. Glowacki
School of Computer Science
Florida International University
Miami, Florida 33199

ABSTRACT

With decreasing cost and size of processors and more sophisticated demands of computer users, it is becoming popular to execute programs in parallel on a distributed network. Subtasks of a program can then be run on separate processors, communicating through shared memory or hard wired links, depending on the hardware and topology of the system.

With ring networks, a token passing or store-and-forward communication scheme is often used. The token passing scheme allows for only one processor to send a message at a time. The store-and-forward scheme allows many messages to travel around the network and must be deadlock free.

This paper presents the simulation of a distributed network of INMOS Transputers. Also discussed are the store-and-forward message passing scheme that is modeled and results obtained thus far.

1. INTRODUCTION

Large computer networks, local area networks, and multiple processor systems are considered to be distributed networks. With these systems, processes of a single program can be distributed over several processors, running in parallel, such that each processor on the network performs a subtask of the main program. Network processors need to share mutual information and are classified as tightly or loosely coupled (Silberschatz and Peterson 1988). Because tightly coupled systems have shared memory, an algorithm must exist to insure mutual exclusive entries into it. Loosely coupled systems have local memory for each processor and communicate by using a message passing scheme.

Processors (nodes) in a ring network are loosely coupled and physically connected in a circle, usually with one-way communication links. Generally, a token or store-and-forward message passing scheme is used to support communication between nodes.

In a token passing scheme, a specific message, the token, continuously circulates through the network. If a node wants to send a message, it must first acquire access to the network by removing the token when it arrives. This sending node forwards a message header followed by the message. When the message has traveled completely around the network, the sending node removes it (guaranteed the destination node received it) and forwards the token. Thus, only one message may travel through the system at one time.

With a store-and-forward message passing scheme, each node has designated storage (buffer) for incoming messages. As messages are received, they are placed in this buffer. When messages can be forwarded, they are removed from it. Because the buffer is a shared resource, the communication scheme is not trivial. The sending and receiving processes form a producer/consumer relationship and special techniques must be employed to prevent deadlock.

With advanced system architecture it is not uncommon to find systems with a large number of processors. The Ethernet local (Ethernet is a registered trademark of the Xerox Corporation) area network, for instance, can support up to 1024 processors (MacDougall 1987). Issues regarding the number of processors required to handle a given work load or system throughput are consistently raised.

Several distributed systems have been simulated in order to evaluate their performance. The maximum mean data rates for several local area networks are presented by Stuck (1983). He explained that transmission medium has a dual purpose: to control access to the network and to transmit the data. Traffic on the network may be of low or high delay. When the network has high delay traffic, it is a bottleneck, and more time may be spent controlling access to the network than actually transmitting data.

Stuck included an evaluation of two ring networks and two bus networks. The ring networks consisted of 100 stations using a token passing scheme. The first had a single station sending to any of the 99 other stations, while the second had all 100 stations sending messages to each other. The bus networks consisted of a token passing scheme and carrier sense multiple access with collision detection. Stuck concluded by stating "Token passing via a ring is the least sensitive to workload, offers short delay under light load, and offers controlled delay under heavy load".

Garcia and Shaw (1986) studied transient behavior of a five-node network using a store-and-forward message passing scheme. Assuming message traffic would be changing

in the future, they were interested in analyzing current communication channels to determine if they were adequate for future loads. In addition they were concerned with how performance might be improved.

Both a sudden burst of messages and a sudden reduction in interarrival time for given periods were modeled. They found network performance severely degraded by these transient message loads.

2. THE REAL NETWORK

The INMOS Corporation manufactures Transputers -- processors specifically designed for parallel processing (Transputer is a registered trademark of the INMOS Corporation). Transputers can be put together as a distributed network connected by fast, hard-wired communication links. Currently, the School of Computer Science at Florida International University has a four-processor distributed network of T414 Transputers.

According to the INMOS Transputer Reference Manual (1987), these processors context switch in a microsecond and perform integer/data moves in approximately 134 nanoseconds. The communication links between processors transmit data at a rate of 10 MHz or 20 MHz (individually switch selectable) with effective rates of .8 and 1.6 million bytes per second, respectively.

The host Transputer, an INMOS B004 board, contains 2M bytes of memory. The network of four Transputers, each with 256K bytes of memory and four bidirectional communication links, resides on an INMOS B003 board. The topology of the network is shown in Figure 1.

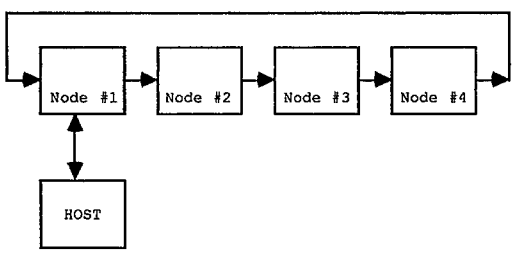

Figure 1: Network Topology

Occam is the native language of the Transputer system. The basic elements of an Occam program are processes that can run sequentially or in parallel. Processes communicate over user-specified logical channels. These channels can be links connecting Transputers or local soft channels connecting processes running on the same Transputer. In addition, Occam supports most of the constructs available in modern high-level languages.

One advantage of the Occam view of processes is they are assigned to processors at compile time. Thus, a program developed as a set of parallel processes on a single Transputer system may be recompiled for any valid Transputer/process mapping (Comfort and Gopal 1988).

A deadlock-free store-and-forward message passing scheme was written by Li Qiang and explained by Qiang, Feild, and Klein (1987a) of Florida International University. The system is comprised of five processes running on each node. Figure 2 displays a single node in the network.

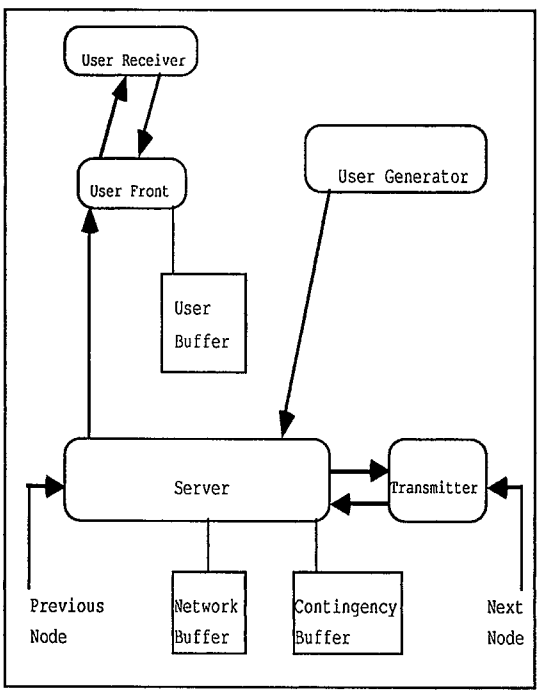

Figure 2: A single node in the network.

There exist two types of processes -- "network" and "local user". Network processes are those that have access to the physical network -- the incoming or outgoing links. Local user processes are those that do not have access to the physical network. There are three local user processes. The main one, performs the application program ("local" to the given node) and generates messages. The second receives all messages for the node. The third acts as an intermediate process supporting communication between the network and the receiving local process.

Each message contains a message header that indicates its source, destination, and length. The header itself is exactly one word regardless of the length of the message. It is important to note that messages are handled at the "word level" -- each word of a message is sent individually although it is part of an entire message.

In order to accommodate incoming messages, there exist three buffers: the user, the network, and the contingency. The user buffer contains those messages received for the local node. The network buffer holds those messages to be transmitted to the next node. The contingency buffer is a protective buffer holding a message that would overflow the network buffer. The contingency buffer is necessary to avoid deadlock and is explained by Qiang (1987b).

The primary responsibilities of the five processes shown in Figure 2 are explained below. To clearly identify each individual process, they have been named and underlined.

The User Generator is responsible for creating messages and passing them over a soft channel to the server. The channel acts as a blocking channel. Therefore, the user generator is blocked between passing each word of a message.

The User Receiver is responsible for reading the messages sent to the current node. It sends a request over a soft channel to the user front to read each word. It is therefore blocked from the time it sends a request until a word is actually forwarded.

The User Front is responsible for the user buffer. It handles the producer/consumer relationship of the server and user receiver. The server passes words to the user buffer via the user front, while the user receiver gets words from the user buffer via the user front.

Occam channels are blocking channels. That is, if process P1 sends a word to process P2, P1 cannot continue until P2 receives the word. If P2 is busy and not ready to receive, then P1 remains blocked. In order to create a non-blocking channel, an intermediate process,P3, must be created (Qiang, Feild, and Klein 1987b).

Accordingly, in order to have the server (P1) pass messages to the local user receiver (P2) without blocking, there must exist the user front (P3) as an intermediate process. The user front takes messages from the server and, transparent to the server, places them in the user buffer. Upon request, it removes them from the buffer and forwards them to the user receiver. Because messages are handled at the word level, a separate request must be issued for each word of the message.

The Server takes words from the incoming link and places them in the appropriate buffer. Messages for the current node are sent to the user front and placed in the user buffer, while all other messages are placed in the network buffer for retransmission. It also receives messages from the user generator and places them in the network buffer for retransmission. Lastly, it answers the transmitter's requests by removing and forwarding messages from the network buffer (one word at a time).

The Transmitter monitors the outgoing link. Whenever the link is available, it requests and receives a word from the server and sends it down the outgoing link.

Deadlock can easily occur in this network if each user generator saturates the network to the point where every node is blocked from servicing the incoming message. In order to prevent this situation, there exists a protocol for filling the network buffer as explained by Qiang, Feild, and Klein (1987a).

In short, the server receives messages from the user generator and the incoming link. It forwards local ones to the user front and fills the network buffer with non-local ones. However, the server places a message from the user generator into the network buffer if, and only if, the entire message can fit. Whenever the network buffer is full, however, the server blocks the user generator and processes messages from the incoming link by filling the contingency buffer. This buffer must be large enough to hold one complete message.

This protocol enables the server to push messages through the system even when the local user process has saturated the system. In other words, if the network buffer fills, the contingency buffer is still available to buffer network traffic.

The term "network buffer" will now refer to both the contingency and network buffers. Physically they are one buffer and logically separated in software. The contingency part is always set to accommodate the largest message size. The network buffer must always be at least as large as two maximum size messages -- one for each part of the buffer.

The Transputer link, like a soft channel, behaves as a blocking link. Therefore, any word sent down a link remains on it until removed by the next node. When traffic is intense, the network can become blocked. It is because of this protocol that the network cannot deadlock.

3. THE SIMULATION MODEL

A comprehensive simulation model was designed to investigate system throughput. Also of interest were the effects of message length variation and network size increases.

Several software packages exist for writing simulation programs. John Comfort at Florida International University has written a distributed simulation package to run on the INMOS Transputer system (Comfort and Gopal 1988). The program identifies objects such as a statistics module, random number generator, and a priority queue handler that can be placed on separate processors of the network.

A simulation program using this package must first instantiate specific instances of these objects. The future events queue is an instance of a priority queue. The objects are then accessed by standard calls. Statistics are updated for an entity in the simulation by sending messages to the

statistics package whenever the entity changes its state.

The servers and entities: In order to simulate the real network it is necessary to determine how processes and messages will be represented. As processes service messages in the real network, servers process entities in the simulation model. Each server must have a set of states and well-defined actions to be performed.

Although processes on the same processor are conceptualized as running in parallel, only one process can actually be running at a time. Thus, for every node in the model, only one server (process) can be servicing (running) at a time. Each type of server had a designated set of states and actions describing the process being modeled and could therefore be in only one state and perform only one action at a time.

Messages in the system: Messages in the real network consisted of two parts: the message header and message body. The header contained the source, destination, and length of the message. In the simulation model, each message header was an entity.

Simulating the buffers: In order to model the user and network buffers that held messages, it was necessary to create two FIFO queues for every node. The queues held the message header entities and local counters were updated to track the total words in a given buffer.

Simulating the links: A Transputer link can only hold one word at a time (message headers are single words). Because actions performed depend on the type of data sent, links were simulated using two variables. The first indicated the type of data on the link: a message header, a word of the message body, or an indication the link was free. If a message header was on the link, then it was necessary to identify the actual entity number. This was done by the second variable.

The Future Events Queue: A single future events queue (FEQ) held the bound event notices for the entire simulation. These notices included scheduling processes to time-out because of waiting for a channel or run time expiring. Also included were notices from a node to another indicating there was data on the link or data was removed. Lastly, there were batch run termination notices, as well as several others.

Simulating the scheduling of processes: Unless a priority scheme for scheduling servers was represented, an unrealistic ordering occurred in the simulation. Therefore, it was necessary not only to keep track of the servers that could process a message, but also the order in which they became available.

For this reason, two queues (Block and Ready) were added to the simulation model. The Block Queue held those servers waiting for some event or condition to occur before they could run, while the Ready Queue held those servers ready to run. The servers in the simulation were placed on the block queue after serving an entity (message) and moved to the ready queue according to pre-defined conditions for the process being modeled. Essentially, this modeled the operating system's scheduler.

System timing: The time needed to perform each action was not easy to determine. The real network communication program was analyzed and it was necessary to literally count instructions (Qiang, Feild, and Klein 1987a). In addition, the INMOS Reference manual was consulted for system timing statistics (INMOS 1987).

Each Transputer cycle takes about 67 nanoseconds -- 15 million cycles per second. In order to acquire accurate results, it was necessary to determine the time needed for each server to perform its action. The level of detail was so crucial that code for each process in the real network communication program was thoroughly evaluated to the point where instructions were literally counted (Qiang, Feild, and Klein 1987a). In addition, the INMOS Reference manual was consulted for system timing statistics (INMOS 1987).

Random number generators: There were five random number streams used for the model. A parameter was sent for each stream indicating the distribution: constant, negative exponential, or uniform. The streams used:

-- Average links a message travels
-- Number of messages to send now
-- Length of the current message
-- Time to run the local user application
-- Operating system delay to schedule a process.

Parameters to the system: There were 16 parameters to the system:

-- The number of batches to run
-- The length of each batch
-- The maximum length of a message
-- The number of messages to send at once
-- The size of the network buffer
-- The size of the user buffer
-- The five means and seeds for the random number streams

4. MODEL VALIDATION

The real network of four nodes was run to acquire comparative results. It was run until each node sent/received 30,000 messages of 15 words to/from the node three links away. This test was run several times with different network buffer sizes but the with user buffer and link speed constant at 2000 words and 10MHz respectively. A few timers were added and the system appeared to reach stability almost immediately.

Intuitively, we could visualize the local user generator flooding the server with

messages so the network buffer would be filled to capacity. Then, the user generator would be blocked and the server would be able to handle incoming messages by placing them in the contingency buffer. At some point, the server could reach a steady state of handling both incoming and local messages.

The simulation was then tested where each node was sending/receiving continuously to the node three links away. The user buffer size and link speed were set to constants of 2000 words and 10MHz respectively. The variant was the size of the network buffer.

This test was run for eight blocks, each representing one second of real time. The network is presumed to have been saturated with messages and reached steady state. The results for blocks three to eight were the same. The average time a message spent in the system for both the real and simulated networks are shown in Table 1 and Figure 3.

Table 1: Simulated versus real: message time in system.

AVERAGE TIME A MESSAGE IS IN THE SYSTEM (SECONDS)

Buffer Size	Real	Simulation	Relative Error
36	.00767	.00492	.3585
54	.00748	.00981	-.3115
150	.03380	.03900	-.1538
300	.08300	.08300	.0000
500	.14616	.14633	-.0012
2000	.60320	.60330	-.0002

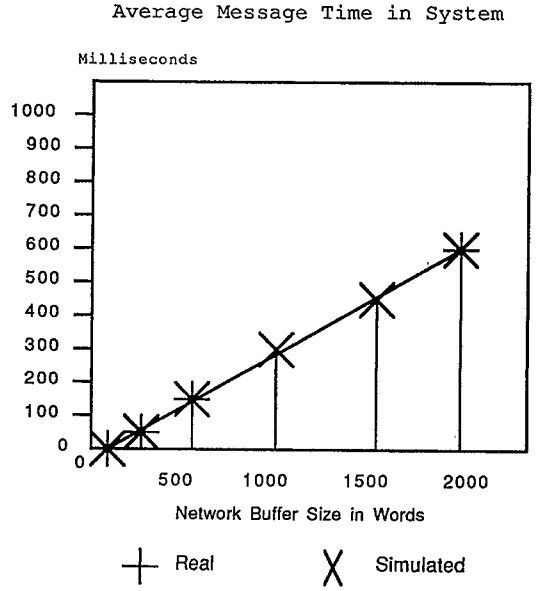

Figure 3: Simulated versus real: message time in system.

When the network buffer is less than 300 words, there was a noticeable difference between real and simulated results. It is possible the cache memory may accommodate these smaller buffers -- with less memory off chip the process can run faster.

The simulation was then run with uniformly distributed random message lengths between 1 and 30 words. Again, each node was sending messages across 3 links at 10 MHz. The results are shown in Table 2 along with the 90% confidence interval which encapsulates the real network's average message time in the system (as shown in Table 1). The user buffer was set to 2000 words. The simulation was set run for 25 intervals each representing one-half second of real time.

Table 2: Four-node network with random message length.

AVERAGE TIME A MESSAGE IS IN THE SYSTEM (SECONDS)

Network Buffer	Average Time in System	Standard Deviation	90% Confidence Interval
36	.00640	.00202	.00308 TO .00972
2000	.79333	.12100	.59426 TO .99235

With several test runs and the results listed here, it was decided the model was valid.

5. FURTHER RESEARCH

The first step of this project was to simulate the real four-node network and verify the results. The simulation model was validated and an extended model has been created. This model was enhanced to simulate networks of sizes greater than four Transputers. Current research includes observing system performance with various message lengths modeling small messages to large file transfers.

Currently, it is not possible to simulate more than 32 Transputers due to the memory constraint. However, we are investigating alternatives to avoid this obstacle. One alternative is to run the simulation on a distributed network itself! However, since each Transputer on the distributed network has only 256 K bytes of memory, the problem may persist. Another alternative is running the model on a VAX 8800 for which Occam is available. Lastly, MicroWay produces the Quadputer which is a four node Transputer network that each Transputer has 1 M bytes of memory.

ACKNOWLEDGEMENTS

I wish to thank my husband, Paul, and Professor John Comfort, whose support and encouragement helped make this project a success. In addition, I give special thanks to Li Qiang and Raja Gopal for sharing their friendship and Transputer expertise.

REFERENCES

Comfort, J.C. and Raja Gopal, R., (1988) "Environment Partitioned Distributed Simulation With Transputers". Proceedings of the 1988 Winter Simulation Conference, 103-108.

Garcia, Albert B., Shaw, Wade H., (1986) "Transient Analysis Of A Store-And-Forward Computer-Communications Network" Proceedings of the 1986 Winter Simulation Conference. Washington, D.C., 752-760.

INMOS Transputer Reference Manual. (1987) INMOS Ltd. 72-TRN 006-03, Bristol, UK.

MacDougall, M.H. (1987) Simulating Computer System Techniques and Tools. The MIT Press, Cambridge.

Silberschatz, Abraham, and Peterson, James L. (1988) Operating System Concepts. Addison Wesley Publishing Company, New York.

Stuck, Bart W. (1983) "Calculating the Maximum Mean Data Rate in Local Area Networks". Computer, May 1983, 72-76.

Qiang, Li, William B. Feild Jr., and Donald Klein. (1987a) "Implementation of a Transputer Ring Network and a Deadlock Prevention Algorithm". Proceedings from the 3rd U.S. Occam User Group Meeting, Chicago, Illinois.

Qiang, Li, William B. Feild Jr., and Donald Klein. (1987b) "Channel Design Primitives in Occam". Proceedings from the 3rd U.S. Occam User Group Meeting, Chicago, Illinois.

AUTHOR'S BIOGRAPHY

JANICE R. GLOWACKI is an associate programmer for IBM in Rochester, Minnesota. She recently received her Master's of Science in Computer Science from Florida International University.

IBM Corporation
Highway 52 & Northwest 37th Street
Rochester, Minnesota 55901
(507)253-4011