Defence Science Journal, Vol 44, No 2, April 1994, pp 113-118
© 1994, DESIDOC

Dynamic Interconnection Techniques for Parallel Communication*

P. K. Das Gupta
Proof and Experimental Establishment, Chandipore, Balasore-756 025

ABSTRACT

Parallel computing has contributed significantly to Defence applications. This field has helped in the development of dedicated hardware/software for ballistic missile defence system. In parallel computers, interprocessor communication mechanism strongly influences overall performance of the system. If a parallel computer is composed of *n* processors and each processor has a performance *p*, then the maximum potential of the system is *np*. The present review concentrates on interprocessor communication. Crossbar and delta network is discussed in detail. Cogent XTM — a better solution over crossbar network is also discussed.

1. INTRODUCTION

Rapid development of VLSI technology has motivated a rich stream of research activity in the field of multiple computer system. Detection and interception of ICBMs, weather forecasting, mechanical modelling, image processing, neural networks, computer vision, and fluid dynamics are some of the application fields of parallel processing.

A multiple computer system includes multiple processors, and these processors may communicate with each other and cooperate at different levels to solve any problem. Communication may take place either by sending messages from one processor to another or by sharing a common memory. It is controlled by one operating system. There are various taxonomies for parallel processing approaches, the field can be divided into two categories: single instruction-multiple data (SIMD) and multiple-instruction-multiple data (MIMD).

The SIMD approach assigns data values to each of the processors and then executes the same series of operations on each of the data values in lockstep. Here, multiple processing elements are supervised by same control unit.

The MIMD approach breaks large processing task into sub-tasks and assigns search sub-task to different processors. An intrinsic MIMD computer is tightly coupled if physical connections are established via memory sharing and microprocessors communicate through a shared main memory. Otherwise, in loosely coupled systems, links and logical communication can be assimilated to packet switching. Each processor has a set of I/O devices and a large local memory where the processor accesses most of its instruction and data.

2. INTERCONNECTION NETWORK

Interconnection network basically refers to communication between processors and the memory modules and between the processors and the I/O systems. The parallel computers can be classified on the internal structure of the interconnected network.

The single shared bus is widely used in parallel systems. Here it is extremely difficult to eliminate bus contention. It can be avoided to some extent by forming the multiple bus network where each processor or memory unit or I/O device is connected to one or more of available buses. The crossbar interconnection network is a multiple bus system, where all units like

Received 18 August 1992
*Invited paper for the Special Issue of DSJ on Computer Applications in Defence (April 1993).

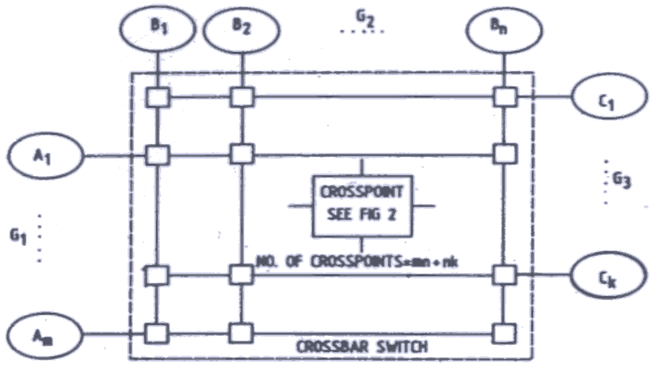


Figure 1. Crossbar interconnection network.

processing elements, memories and I/O devices are interlinked as shown in Fig. 1. Other popular interconnected networks include tree and hypercube structures. Tree type structure is suited to certain special kind of parallel processing. Where neighbouring processors can communicate quite fast, but communication between non-neighbouring processors is slower and it requires intermediate processors to store and forward message transfer. In *n* dimension hypercube interconnection network 2^n processors are used and each processor is connected to *n* neighbours. The hypercube structure has shown suitability to fairly wide range of programming tasks. However, relatively very few hypercube structures are implemented commercially.

The single bus, multiple bus and crossbar interconnections are dynamic interconnection structures. whereas tree and hypercube are static interconnection structures. This article discusses crossbar switch and its modifications in detail, and a solution proposed by the Cogent Research in Beaverton, USA.

3. CROSSBAR CONNECTION

In nonblocking crossbar, there is separate path available to each memory unit, as shown in Fig. 1. It is used to connect three groups of units $G_1=\{A_1, A_2, \ldots, A_m\}$, $G_2=\{B_1, B_2, \ldots, B_m\}$ and $G_3=\{C_1, C_2, \ldots, C_m\}$ so that any unit of G_1 can be connected to any unit of G_2 or G_3 but two units in the same group need not be connected. For example, G_1 could be a set of processors, G_2 could be a set of memory modules and G_3 could be a set of I/O devices. In crossbar connection each cross point is capable of switching parallel transmission and resolving multiple requests for access

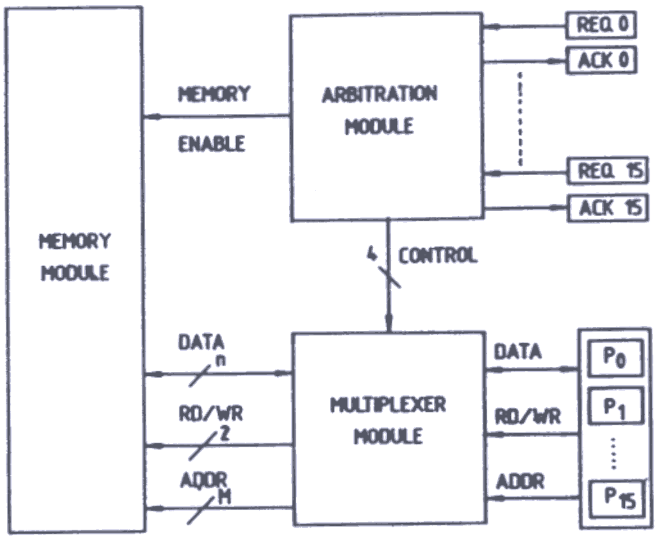


Figure 2. Crosspoint functional structure in crossbar network.

to the same memory module occurring during a single memory cycle. These conflicting requests are handled on a predetermined priority basis.

In the following discussion, the example of *C.mmp* (Fig. 2) to implement the functional structure of a crosspoint in a crossbar network is considered.

Step I : Each memory module request signal (REQ) to arbitration unit.

Step II : Processor with the highest priority is being selected. Priority encoder decides the selection

Step III : Arbitration module returns an acknowledge signal (ACK) to the related processor.

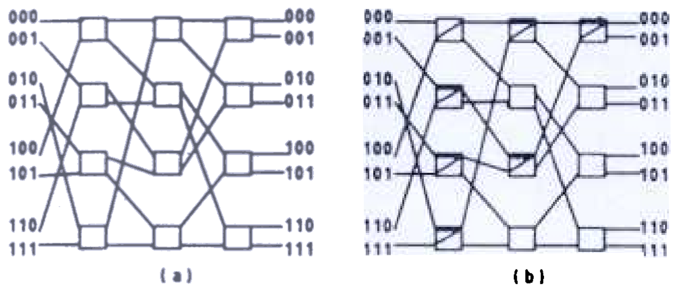
Step IV : Selected processor initiates its memory operation, and

Step V : Multiplexer module multiplexes data addresses of words within the module using a 16-to-1 multiplexer. The multiplexer is controlled by the encoded number of the selected processor, which was generated by priority encoded with the arbitration module.

Although crossbar networks have been employed by a few computer systems, their hardware complexity quickly becomes prohibitive as the number of processors, memory modules and I/O devices increase. One of other major difficulties with the crossbar is the rapid growth rate of the number of connections that must be made when new processors, memories and I/O devices are added. Crossbar has the complexity of n^2 connections, where n is the number of switches in crossbar network. High cost of crossbar switches prevents the growth of crossbar network.

4. DELTA NETWORK

General $a^n \times b^n$ delta network consists of a^n sources and b^n destinations, n number of stages and the ith stage has $a^{n-1}\ b^{n-1}$ crossbar modules of size $a \times b$. We will explain data network by means of three stage $2^3 \times 2^3$ switching network (Fig. 3(a)). The processor pairs are connected to each other either with T (through) or X (cross) switch states which are dynamically set by control logic associated with interconnected network. Delta network possesses full access property since it is possible to connect every processor (P_i) to the other (P_j). An example is shown to connect two processors in Fig. 3(b). Figure 4 shows one state of delta network. In delta network it is not possible to establish a connection between any pair of processors that are not using the network, without altering the switch arrangements already established to link other processor, resulting in communication delay due to nonblocking property.



SWITCH SETTING OF DELTA NETWORK TO CONNECT 000 TO OTHER PROCESSOR

STATE OF SWITCHING ELEMENT CAN BE T (THROUGH) OR X (CROSS)

DESTINATION	SWITCH SETTING
001	TTX
010	TXT
011	TXX
100	XTT
101	XXX
110	XTX
111	XXT

Figure 3. three-state $2^3 \times 2^3$ delta switching network

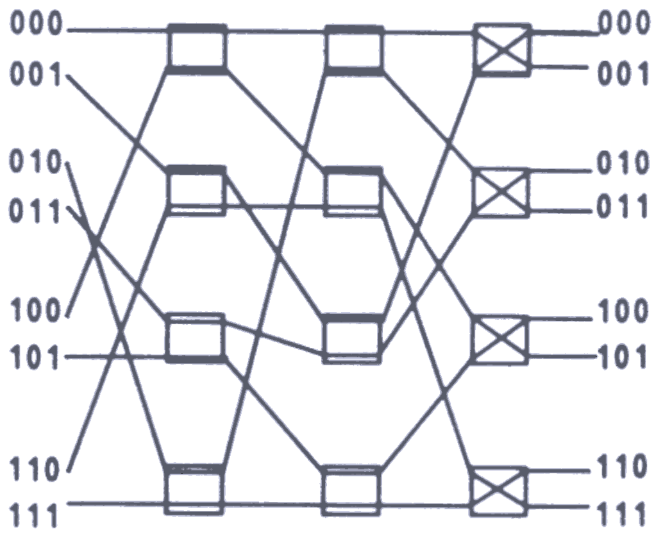


Figure 4. One state of delta network

5. PERFORMANCE COMPARISON OF CROSSBAR AND DELTA NETWORKS

The assumptions for the analysis are :

(a) Each processor generates random and independent request for a word in memory and request are uniformly distributed over all memory modules.

(b) At the beginning of every cycle, each processor generates a new request with a probability r, and

(c) The requests which are blocked, i.e., not accepted, are ignored.

5.1 Crossbar Network

Consider $a \times b$ crossbar networks with a processor (source) and b memory (destination). We will analyse the networks for finding the expected bandwidth. Bandwidth is expressed as the average number of memory requests accepted per cycle.

Let r be the probability that a processor generates a request during a cycle and $q(i)$ be the probability that i requests arrive during one cycle. Then

$$q(i) = {}^aB_i.\ r^i.\ \cdot r)^{a-i} \qquad (1)$$

where ${}^aB_i$ is the binomial coefficient

Let $E(i)$ be the expected number of requests accepted by $a \times b$ crossbar during a cycle, given that i requests arrives in the cycle.

Probability of a particular module requested is

$$(b^i\text{-}(b\text{-}1)^i)/b^i$$

Then expected number of acceptance, given in i request is

$$E(i)=((b^i-(b-1)^i)/b^i).b)$$

$$=((1-(b-1)/b)^i.b) \qquad (2)$$

Thus the expected bandwidth $B(a,b)$, that is, the average number of requests accepted per cycle is

$$B(a,b),= \Sigma\ E(i)q(i);\ 0 \leq i \leq p$$

$$= b-b(1-r/b)^a \qquad (3)$$

Let P_A be the probability that an arbitrary request will be accepted, than

$$P_A=B(\mathrm{a,b})/ra = (b/ra)-(b/ra)(1-r/b)^a \qquad (4)$$

5.2 Delta Network

Consider delta network of size $a^n \times b^n$ made of $a \times b$ crossbar modules. Clearly a^n processors are connected to b^n memory modules. We can apply results of Eqn. (3) of $a \times b$ crossbar network since our assumption of independent request holds good for $a \times b$ module in delta network. Thus dividing Eqn. (3) by b (number of output lines of $a \times b$ module) gives the rate of requests on any one of b output lines, thus we get

$$1-(1-r/b)^a \qquad (5)$$

For any stage of delta network, r_{out} (output rate of request) is a function of r_{in} (input rate), thus

$$r_{\mathrm{out}}\ :(1-r_{\mathrm{in}}/b)^a \qquad (6)$$

In delta network we have to recursively evaluate the output rate of any stage starting at stage 1. The output rate of the final stage n determines the bandwidth of delta network.

Let r_i be the rate of requests on an output line of stage i. Then $B(a^n,b^n)$ of an $a^n \times b^n$ delta network is given by

$$B(a^n,b^n) = b^n.r_n \qquad (7)$$

where $r_i=1-(1-(r_{i-1}/b))^a$ and $r_o=r$

The probability that a request will be accepted *is*

$$P_A=b^n r_n/a^n r \qquad (8)$$

Table 1. Probability of acceptance of crossbar and delta networks

No. of processors	Acceptance probability	
	Crossbar network	Delta network
	1.00	1.00
2	0.82	0.93
4	0.70	0.82
8	0.67	0.60
16	0.64	0.41
32	0.63	0.40
64	0:62	0.39
128	0.61	0.37
256	0.61	0.30

Table 1 shows the probability of acceptance P_A for $2^n \times 2^n$ and $a \times a$ crossbar networks. It is clear from the table that the curve in crossbar approaches a constant value whereas in delta network it continues to fall as number of processors grows.

6. THE COGENT

In a crossbar network, all the processors can share a common communication bus; this ties up the bus if two processors have a lot of data exchange bringing communication for the rest of the system to a screeching halt. To overcome this problem Cogent Research in Beaverton, Oregon, USA suggested the Cogent's desktop supercomputer, the XTM. Cogent's new desktop supercomputer, the XTM can connect any number of parallel processors without passing data hand-to-hand through a network or tying up a common bus when there is lot of data to exchange. Instead, the Cogent machine has a hybrid communication architecture that has both a common bus and unique network system.

Suppose processor A wants to send a large data to processor B. If A sends data through the common bus, it would tie up the bus as long as data communication takes place between A and B, and this will lead to communication bottleneck. Instead, A requests switch for a direct connection to B. Upon request, the intelligent crossbar switch can directly connect any two processors in the network. Consequently any two

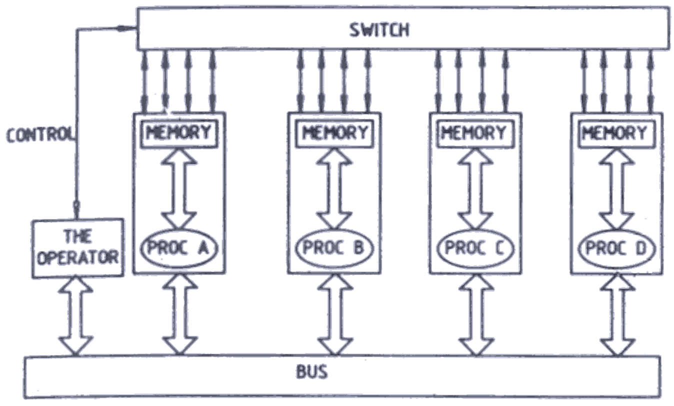


Figure 5. The Cogent XTM communication system

processors can talk either through the shared bus or through a temporary direct connection (Fig. 5). Once the connection is made, *A* can send data to *B* at high speed without interfering with any other processor's communications. After data transfer is over, *A* sends another message to switch controller, asking it to disconnect *A* from *B*, and the two processors are free to make new connections. Similarly, every other pair of processors can be connected in the same manner. While *A* and *B* are exchanging data, *C* and *D* can make their own connection.

By this method, one thousand processors can communicate using only 4000 serial lines which is less than 1 per cent of the number required for a fully connected network. The XTM's operating system is based on the Linda parallel-programming concept (see Appendix), which is effectively blind to the number of processors in the computer. A program written in FORTRAN or C, using a Linda extensions will run on a minimal XTM system. Add two (or dozen) more processors, and the program will run the exactly the same way—but nearly twice (or a dozen times) faster.

CONCLUSION

Parallel computers have been designed around a variety of inter processor communication networks. Shared bus is the easiest to implement and is most common in smaller parallel computers. Due to rapid advancement in VLSI technology, it has become feasible to construct massively parallel systems based on static interconnection structures as meshes, trees and hypercubes. Another class of parallel computers includes crossbar, multistage switching networks such as delta network etc. These mechanisms exhibit various trade-offs between processor throughput, communication delays, and the programming complexity.

BIBLIOGRAPHY

Briogs, F.A. & Dubois, M. Effectiveness of private caches in multiprocessor systems, with parallel-pipelined memories. *IEEE Trans. Computers*, 1983, 48-59.

2 Cantoni, Virgino; Ferretti, Marco & Lombardi, Luca. A comparison of homogeneous hierarchical interconnection structures. *Proceedings of the IEEE*, 1991, **79** 416-28.

3. Feng, T.Y. *IEEE Trans. Computers*, 1981, 12-27.

4. Patêl, J.H. Performance of processor-memory interconnections for multiprocessors. *IEEE Trans. Computers*, 1981, 771-80.

5 Enslow, P.H. Multiprocessor organisation — a survey. *ACM Computing Surveys*, 1977, 103-29.

6. Gelernter, D. Getting the Job done ("Linda" a parallel programming language). *Byte*, 1988, **13** 301-08.

Hayes, Frank. The crossbar connection (Cogent XTM). *Byte* 1988, **13**, 278-79.

8. Hayes, J.P. Computer architecture and organisation. McGraw-Hill New York, pp. 57-80, 468-77, 649-60.

9 Hwang, Kai & Briggs, F.A. Computer architecture and parallel processing. McGraw-Hill, New York. pp. 1-49, 459-508.

10. Desrochers, G.R. Principles of parallel and multiprocessing. McGraw-Hill, New York. pp. 189-96.

APPENDIX

PARALLEL PROGRAMMING LANGUAGE 'LINDA

Linda is an efficient, portable and easy to use parallel programming language. Linda consists of a few simple operations that embody the tuple space (TS). TS is a bag of tuples, where a tuple is simply sequence of typed field. Linda provides operators for dropping tuples into the bag. hauling tuples out, and reading these without removing them. To locate the one combination of field, values are to be searched since tuples do not have addresses. Linda also supports live tuples. When a live tuple is dropped in it, it is envaluated in parallel with the task dropped in. So, to create *n* parallel processes simply drop *n* live tuples. There are four basic TS operation :

out (t)—causes the tuple *t* to be added to TS, the executing process continues immediately;

in (s)—causes some tuple *t* that matches the template *s* to be withdrawn from TS, the value of the actuals in *t* are as signed to the formals in *s*, and the execution process continues. In case more than one matching *t s* are available, one is arbitrarily chosen. If no match is being found, process suspends until one is available;

rd (s)—same as *in (s)*, with actuals assigned to formals as before and matched tuple remains in TS; and

eγal (t)—same as *out (t)*, except *t* is evaluated after it enters TS; *eval* implicitly forks a new process to perform the evaluation.

Linda is suitable for fast database access, expert database system; for expert monitoring system, etc. Linda seems to contribute a lot, to the new era of parallel software.