NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us a poor photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de mauvaise qualité.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE

DCA -
A DISTRIBUTED COMMUNICATION ARCHITECTURE

by

Hon W. Wong

Submitted to the School of Graduate Studies
in partial fulfillment of the requirements for
the degree of Master of Applied Science

Department of Electrical Engineering
Faculty of Science and Engineering
University of Ottawa
Ottawa, Ontario

February 1981

ABSTRACT

This thesis (1) examines the concept of a distributed processor, as viewed from the communications architecture point of view, (2) proposes a new distributed communication architecture, DCA, and (3) suggests an interprocess communication mechanism within a processor and between processors.

The memory interference effect is examined, on the time-shared bus and crossbar switch systems. The analysis serves as the basis of performance on CPU utilization under memory contention situation.

DCA, a distributed communication architecture, is proposed in this thesis. The architecture can be used to simulate operations in different classes of machines, as proposed by Flynn[6]. DCA can also be used as a local area network composed of independent host computers. The reliability, expansion potential and reconfiguration capability of DCA are also discussed.

Finally, the requirements of a process-to-process communication protocol are examined. An uniform interprocess communication mechanism is suggested to allow local and remote process-to-process communication.

ACKNOWLEDGMENTS

I wish to thank my supervisors, Dr. M. Goldberg and Dr. N.D. Georganas, for their patience, support and understanding throughout the preparation of this thesis.

I also like to thank Dr. C.V.W. Armstrong for his guidance and support during the first year of my graduate study.

My final thanks go to my family and friends, especially Mr. Harris Au, Mr. Anthony Ma and Mr. K.B.Fung for their encouragement and support during the course of my graduate study.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1 - INTRODUCTION

The concept of a distributed processor, as viewed from the communications architecture point of view, has been explored by Enslow[1], Thurber[2] and Anderson and Jensen[3]. In his definition of a "distributed" data processing system, Enslow provides a three dimensional view to characterize the degree of decentralization of a distributed system. Enslow claims that a distributed processor must have distributed hardware, a distributed data base, and distribution of control. Thurber chooses a hierarchial, software system-oriented model, which contains multilayered protocols, e.g., user-level protocol, interprocess communication protocol and intercomputer communication protocol. Anderson and Jensen view distributed processors as models characterized by transfer strategy, transfer control method, transfer path structure and system architecture.

The motivations for distributed systems are many. Some of them are improved throughput and response time, modularity, load leveling, high reliability, easy expansion, configuration flexibility, and resource sharing[1][4]. Recent advances in semiconductor technology reduce the cost of processors and increase decentralized processing through the distribution of processing functions to these processors. Their moderate throughput, coupled with the need for resource sharing and communication, attract more attention to their interconnection mechanisms.

This thesis proposes the design of a distributed processor, as viewed from the communications architecture point of view, and suggests an interprocess communication protocol for local and remote processes communication.

The communication structure is considered to be comprised of paths and switches. From a processor interconnection standpoint, memories are also considered as paths[4]. Paths are the media over which information is transferred. They may be common or dedicated links (busses), or common memories. Switches are devices used to control the transfer of information. They may be hardware switches, e.g., crossbar switches, or software switches, e.g., computers such as the ARPANET IMPs[4].

One of the common interconnection structures between processors is common memories in a time-shared bus (common bus) or a crossbar switch configuration[5]. In order to understand the effect of memory interference (access conflicts) on throughput of these systems, an analysis is made in the next chapter.

A brief description of the classification of computing systems by Flynn[6] is also included in the next chapter. The use of the interconnection system of Distributed Communication Architecture (DCA) in emulating the modes of operations defined by Flynn is demonstrated in later chapters.

Chapter 3 explains the proposed DCA structure and its merits, followed by two application examples.

Chapter 4 suggests an interprocess communication protocol for intra-module and inter-module process communication. The distributed operating system and distributed data base issues, however, are not discussed in this thesis, due to their diverse scope.

CHAPTER 2 - INTERCONNECTION SYSTEMS

In this chapter, an analysis is made on the effect of memory interference in a time-shared bus and a crossbar switch interconnection system. A brief description of the classification of computing systems by Flynn[6] is also made. The emulation of their modes of operation is demonstrated. It will be shown in the next chapter that the DCA's interconnection system can be used to emulate the modes of operation defined by Flynn.

2.1 Time-shared Bus.

An analysis of the effect of memory interference in a time-shared bus interconnection system was made by Ravindran and Thomas[7]. The analysis was verified with a software simulation which proved the analysis to be very accurate. The processing time, t(p), after a memory access, is assumed to be greater than or equal to the memory rewrite time, t(w). If t(w)=0, the maximum number of memory access is therefore limited by 1/t(a), where t(a) is the memory access time. With t(p) less than t(w), a queueing model cannot be used since the arrival rate can be greater than the service rate, thus resulting in infinite delay. Moreover, the case t(p)=t(w) serves as a boundary condition for the other two cases[8].

The time-shared bus model assumes a single memory unit and multiple processors sharing the bus. The first parameter required for the calculation is the probability of a memory reference during any CPU minor cycle time. Let this probability be denoted by R.

This probability can be statistically estimated by taking a large sample of an application program. At first, the frequency of execution, with respect to a set of instruction classes, is collected. Secondly, from the manufacturer's specification, the frequency of memory references for a particular instruction class can be found. The product of the frequency of execution and the frequency of memory references gives the memory utilization factor for that instruction class. The sum of all the instruction classes memory utilization factors gives the mean average percentage of a memory reference.

Let P be a fixed probability vector denoting the probabilities that 1,2,...,S processors are working, i.e., not waiting for service from memory. A processor is 'working' if it is being serviced by memory at that instant or the processor is not accessing memory at that time. For example, if there are 10 processors 'working' at an instant, it means there are 10 processors working within the CPU and not accessing memory, OR there are 9 processors working within the CPU and not accessing memory, together with 1 processor being serviced by memory at that instant. In other words, a processor is 'not working' if it is waiting for service from memory. Using the Markov chain model PA=P, or P(t)A(t)=P(t+1),

$$\begin{pmatrix} P_1 & P_2 & \cdots & P_S \end{pmatrix} \begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1S} \\ A_{21} & A_{22} & \cdots & A_{2S} \\ \cdots & \cdots & \cdots & \cdots \\ A_{S1} & A_{S2} & \cdots & A_{SS} \end{pmatrix} = \begin{pmatrix} P_1 & P_2 & \cdots & P_S \end{pmatrix}$$

or, $P_i = P_1A_{1i} + P_2A_{2i} + \ldots + P_SA_{Si}$

or, the probability that the system is in state i

$= \sum_{j=1}^{S}$ (probability that the system is in state j) x (probability that the system transits from state j to state i).

The state, in this case, means the number of processors 'working'. The set of simultaneous equations, PA = P, can be solved to find the vector P ; together with the condition that $P_1 + P_2 + \ldots + P_S = 1$.

The transition matrix, A, is modeled as sequences of Bernoulli trials. For example, out of M processors working at the present memory cycle, the transition probability that N processors working at the next memory cycle is

$$A(M,N) = \binom{M}{M-(N-1)} R^{M-(N-1)} (1-R)^{N-1} \quad \text{for } M \geq N;$$

$$A(M,N) = \binom{S}{1} R(1-R)^{S-1} + (1-R)^S \quad \text{for } M=N=S;$$

$A(M,N) = (1-R)^M$ for $N=M+1$;

$= 0$ for $N > M+1$, since there can only be one processor serviced by memory in one memory cycle and return to 'working' state at the end of the cycle.

The probability that memory is idle is $= P_S(1-R)^S$ or S processors are 'working' and none of them is being serviced by memory at that interval.

The expected value of the CPU throughput, E(CPU), is

$= \sum_{i=1}^{S} i \times P_i$

For the number of processors S = 3, Table 1 shows the probability vector, the memory utilization percentage and the expected value of the CPU throughput as calculated against different memory reference probabilities.

Table 1

R(memref)	P(1)	P(2)	P(3)	Memutil%	E(CPU)
0.1	0.0014	0.0334	0.9651	29	2.96
0.2	0.0153	0.1376	0.8468	56	2.83
0.3	0.0620	0.2869	0.6508	77	2.59
0.4	0.1566	0.4168	0.4262	90	2.27
0.5	0.2941	0.4706	0.2353	97	1.94
0.6	0.4530	0.4384	0.1083	99	1.65
0.7	0.6127	0.3472	0.0399	99	1.43
0.8	0.7602	0.2294	0.0102	99	1.25
0.9	0.8899	0.1089	0.0011	99	1.11

For the number of processors S = 6, Table 2 shows the memory utilization percentage and the expected value of the CPU throughput as calculated against different memory reference probabilities.

Table 2

R(memref)	Memutil%	E(CPU)
0.1	57	5.75
0.2	92	4.63
0.3	99	3.32
0.4	99	2.50
0.5	99	2.00
0.6	99	1.67
0.7	99	1.43
0.8	99	1.25
0.9	99	1.11

A graphical description of memory reference probability, memory utilization percentage, and CPU utilization for 3 and 6 processors is shown in figure 1. The graph shows that when the memory reference rate approaches to 1, and in both cases, the CPU utilization approaches to 1.

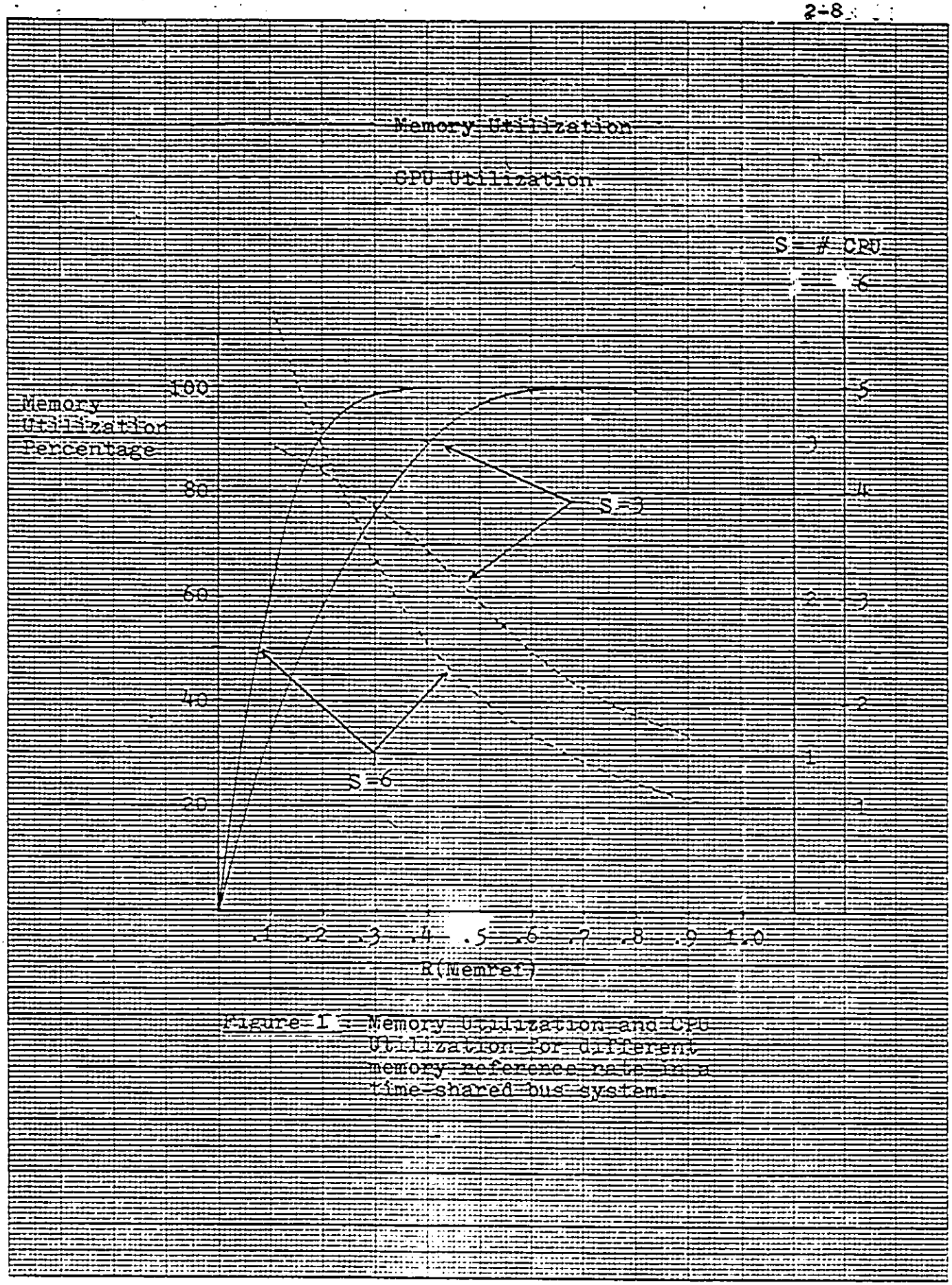


Figure 1 - Memory Utilization and CPU Utilization for different memory reference rate in a time-shared bus system.

2.2 Crossbar Switch.

The analysis of memory interference in multiprocessor systems with a crossbar switch interconnection system was made by Bhandarkar and Fuller[8][9]. Bhandarkar analyzed two cases; one with $t(p)=t(w)$, and another with $t(p)> t(w)$. If $t(w)=0$, in the absence of memory contention, the theoretical maximum value of the average number of busy memory modules is $m * (t(a)/(t(a)+t(p)))$ [8]; where m is the number of memory modules. This serves as the upper bound condition. Systems with $t(p) > t(w)$ are generally processor speed limited and relatively little performance improvement will be obtained if the memory speed is increased.[8] With $t(p) < t(w)$, the queueing model cannot be used.[8] In this section, the case $t(p)=t(w)$ is discussed because the analysis serves as a boundary condition for the other two cases. It is shown in the following, that as multiprocessor systems grow to include more processors, we can expect half of the processors to be active.[9]

For a multiprocessor system consisting of n processors and m memory modules, the probability that one memory bank is idle is equal to

(number of ways of assigning n processors to m-1 memory modules)/ (number of ways of assigning n processors to m memory modules). Using the results of Jackson[10] and Gordon and Newell[11], the probability that anyone of the memory modules is idle =

$$\binom{n+m-2}{m-2} \Big/ \binom{n+m-1}{m-1} = 1 - \frac{n}{n+m-1}$$

The expected number of busy memory modules =

$$\sum_{i=1}^{m} \text{Probability } (m(i) \text{ is busy}) = \frac{mn}{m+n-1}$$

For m = n, the result is $n/(2-1/n)$. As n approaches to infinity, the limit of (expected number of busy memory modules) = n/2. The important conclusion states that as multiprocessor systems grow to include more processors, we are not faced with a law of diminishing returns. If we have the same number of memory modules, we can expect half of the processors to be active[9].

The processor-memory relationship with respect to the expected number of busy memory modules and the expected CPU utilization is shown in figure 2. As the number of processors and memory modules increases, the average CPU utilization approaches 50%. In a 2 by 2 configuration, the CPU utilization is only 35%. Thus, the crossbar switch interconnection system yields a better utilization percentage as the number of processors and memory modules grows. The utilization percentage levels off to about 50% and remains at that level no matter how many processors and memory modules are added. This interconnection system is best for moderate to large systems. One important factor in using the crossbar switch system is the cost, as the number of crosspoint logic increases as n^2.

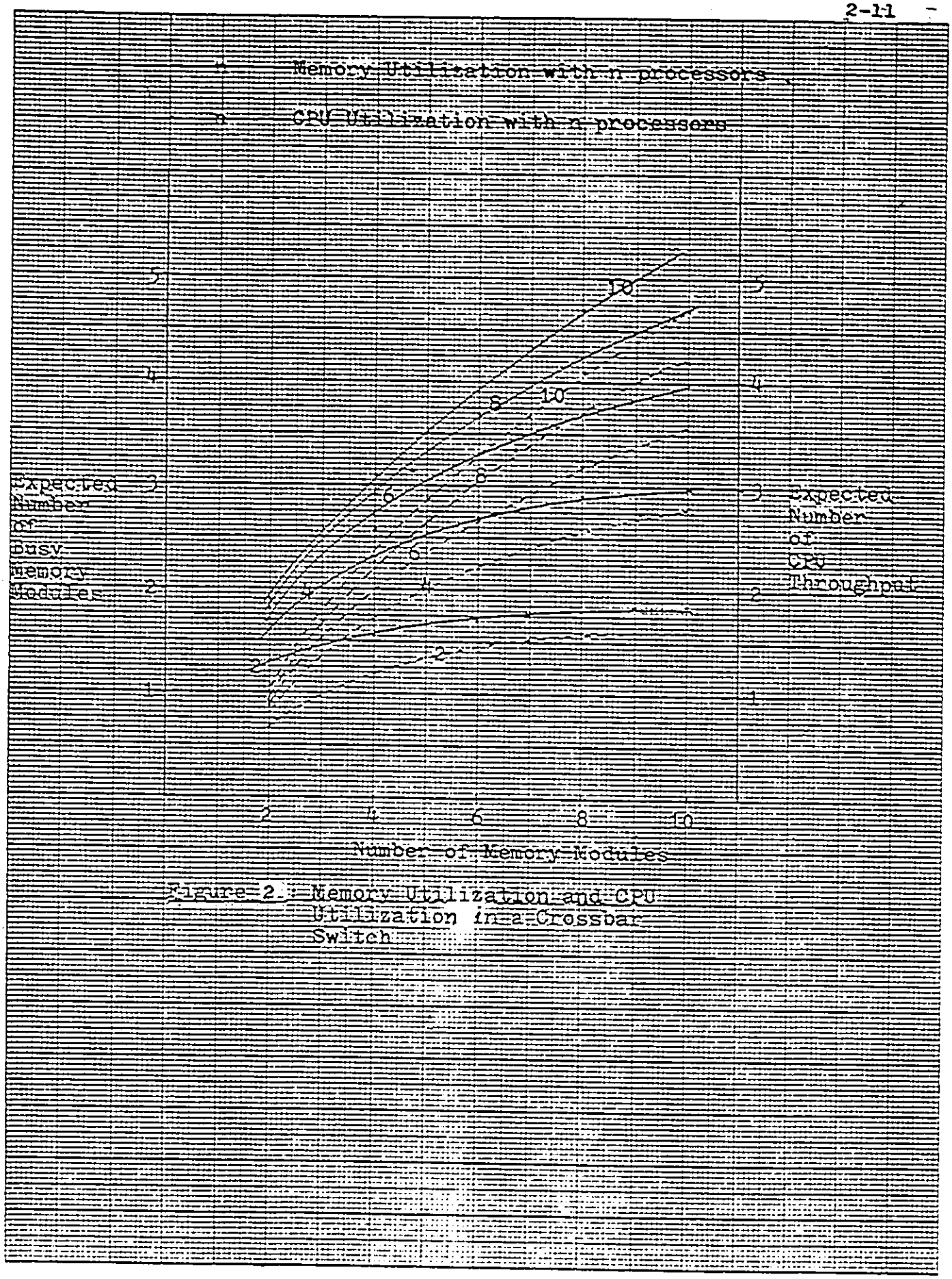


Figure 2.: Memory Utilization and CPU Utilization in a Crossbar Switch

2.3 Emulation of Interconnection Networks.

An interconnection network can be used to simulate a wide variety of other interconnection networks. Smith and Siegel[12] use an emulator network to simulate single stage and multi-stage SIMD interconnection networks which include the STARAN, the Omega, the data manipulator, and the Illiac networks[12]. The SIMD (Single-Instruction-stream, Multiple-Data-stream) organization is one of the four classifications in Flynn's definition of computing systems[6]. To examine the interconnection systems of Flynn's classifications, the four main structures are described briefly below:

(1) SISD: Single-Instruction-stream, Single-Data-stream organization consists of a single control unit, a single processing unit and a single data path from the processing unit to the memory unit. Machines such as IBM S/360, DEC PDP-11, and most conventional computing equipment available today belong to this class.

(2) SIMD: Single-Instruction-stream, Multiple-Data-stream organization consists of a single control unit, and multiple processing units with a single data path from each unit to memory. Array vector computers or associative processors such as Illiac IV and Solomon belong to this class.

(3) MISD: Multiple-Instruction-stream, Single-Data-stream organization consists of multiple control units and processing units; a single data path from memory to one processing unit and a single data path to memory from another processing unit. This organization can also consist of only one control

unit but this unit generates the same control signal to all processing units. This class of machines is sometimes also called pipelined processors. Machines such as Texas Instrument's Advanced Scientific Computer belong to this class.

(4) MIMD: Multiple-Instruction-stream, Multiple-Data-stream organization consists of multiple control units, and multiple processing units with a single data path from each unit to memory. This organization is also called multiprocessor. Machines such as Burroughs D825 and Ramo-Wooldridge RW-400 belong to this class.

Figure 3 shows the classification of computing systems by Flynn. The two attributes in the classification are the control signals (instructions) and data. These modes of operation can be emulated by applying different or the same control signals and data within the interconnection system of a distributed processor. The control signals can be in the form of operation code passed via a communication link to all or a subset of processing elements. Thus, a broadcast operation is sometimes necessary. Data can also be transported along different communication links to the processing elements and different or the same control sequences can be applied to different data stream. The DCA interconnection system, as shown in a later chapter, is well suited for use in emulating these kind of operations. It can be used to study the effectiveness of various design parameters in a prototype before the actual system is built.

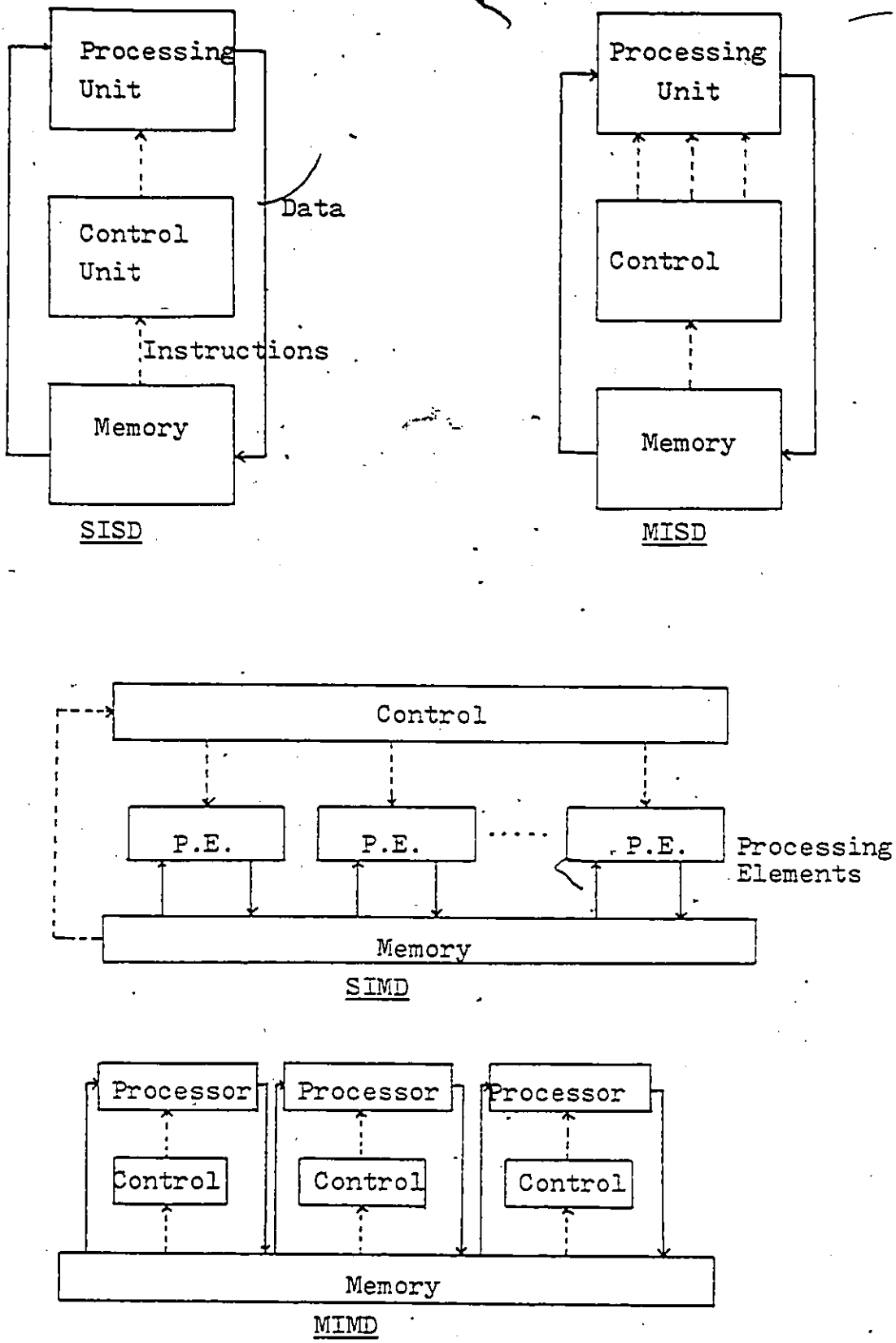


Figure 3

Classification of Computing Systems by Flynn

CHAPTER 3 - DCA, A DISTRIBUTED COMMUNICATION ARCHITECTURE

3.0 DCA, A Distributed Communication Architecture.

The design of a distributed processor, as view from the communications architecture point of view, is proposed in this chapter. DCA, a distributed communication architecture, is shown in Figure 4.

Distributed systems, composed of an interconnection of processing elements, communicate via common memory, a bus or a communication line. Their differences are judged by the degree to which their attributes match an environment. Their attributes are (1) interdependency, (2) extent of resource sharing, (3) degree of autonomy, (4) method of connection and (5) the complexity of each processing element. A multiprocessor can be considered as a distributed system and a network of multiprocessors joined together by some means can also be considered as a distributed system. DCA makes use of the powerful throughput of a multiprocessor together with flexibility, resource sharing and reliability of a network of multiprocessors.

DCA is made up of a number of computer modules (CM) and cables running parallel along the x- and y- axis. Each computer module is connected to one cable running in the x- axis and another cable running in the y- axis. The number of computer modules and the number of cables are determined by the application. It is the purpose of this section to show the versatility of this structure.

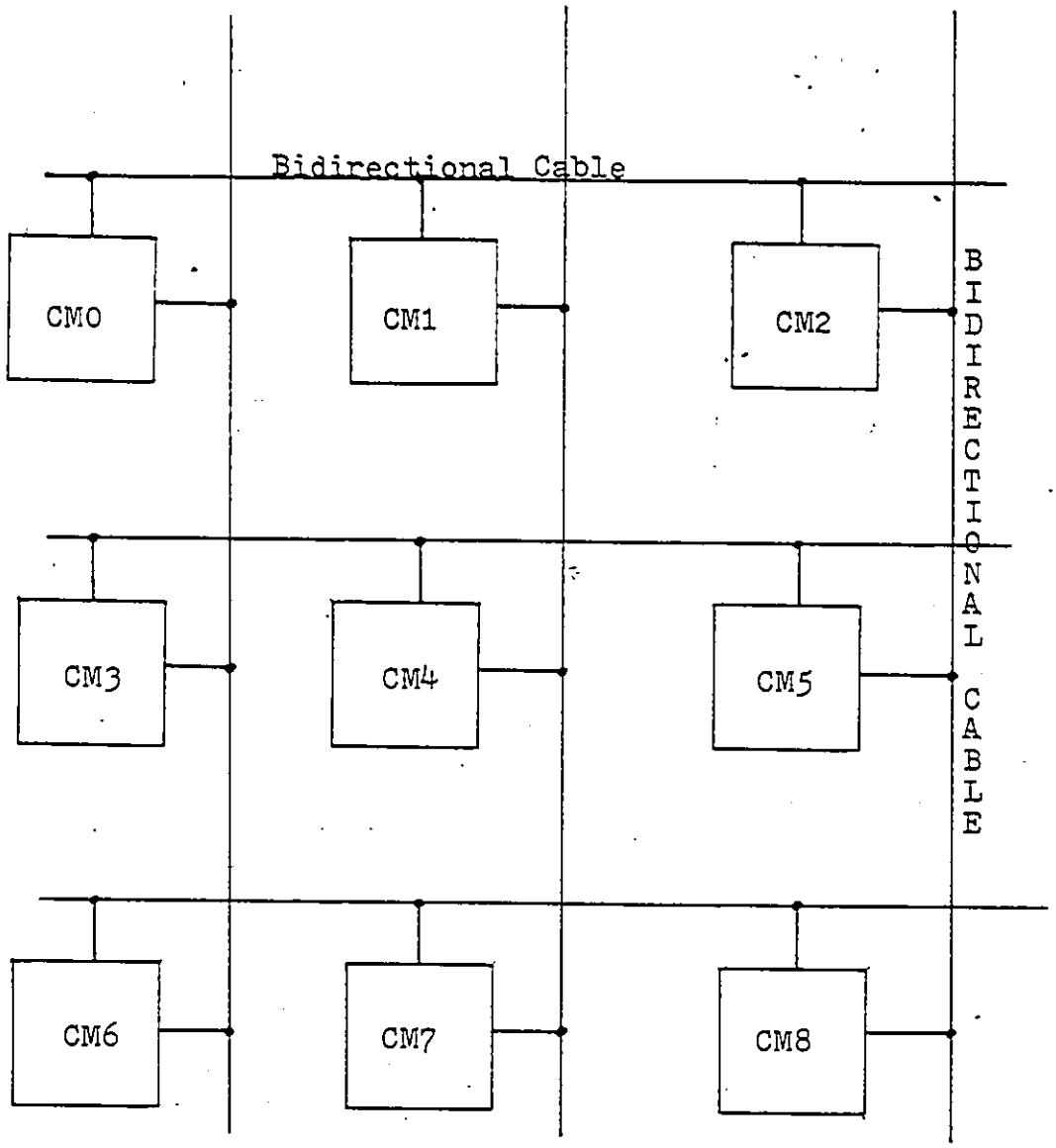


Figure 4: DCA, Distributed Communication Architecture

3.1 Computer Module (CM).

The interconnection system of a computer module is shown in Figure 5. It is made up of n processors and n memory modules interconnected by a crossbar switch. The value of n ranges from 1 to 16. Patel showed that the performance/cost of a crossbar switch is best provided, if the number of processors is not over 16 (33). The proposed processor is an Intel 8086. It contains a Bus Interface Unit (BIU) which can pre-fetch instructions and save them in a queue for up to 6 instruction bytes - just enough space to accomodate a 'mov memory,accumulator' instruction together with a 'mov accumulator,memory' instruction. The address space of the 8086 allows access of one megabytes of memory. The XCHG and LOCK instructions in the 8086 can be used to control access to common resources. The XCHG instruction can obtain the current value of a semaphore and set it to busy in a single instruction. Since the XCHG instruction requires two bus cycles, it is necessary to precede the XCHG instruction with a LOCK instruction which establishes control over the bus throughout execution of the instruction that follows the LOCK instruction.

The 8086 allows different addressing modes: direct addressing, register indirect addressing, based addressing, indexed addressing, based index addressing, string addressing and I/O port addressing. There are development tools such as ISIS-II operating system, PLM-86 high level language, ASM-86 assembler etc. to support the development process.

The memory system is n-way interleaving providing simul-

taneous access for all processors. It is divided into program store and data store; leaving a possible inclusion of cache memory in the future. Memory units are protected with additional bits for error detection and correction. How many bits are required depends on the application; cost versus how much reliability. It should be pointed out here that local program store and data store can be provided to any individual processor, if the application warrants it.

Interprocessor interrupts are performed in hardware. A programmable interrupt controller (e.g. Intel 8259A) in each processor handles interrupts from other processors and I/O devices. An interrupt can be used to synchronize processes in different processors. Since the processors share common memory, software techniques can also be used for this purpose.

I/O controllers share the main memory with other processors. Devices are connected to their controllers via another crossbar switch. This allows resource sharing for all processors, and eliminates any request to a particular processor for I/O services, except in intermodule communication through the cable. This configuration increases the reliability and flexibility.

There are two Ethernet-type control units associated with each computer module. One control unit is connected to the 1st processor and the other to the Nth processor. The control unit performs the Ethernet protocol (as explained in a later section) and error checking and recovery. One control unit taps into a cable running parallel to the x- axis and the

other control unit taps into another cable running parallel to the y- axis. The cables can be of different medium: fiber optic, coaxial or twisted-pair. The application determines the communication requirements of each cable, and the DCA structure is flexible to different applications.

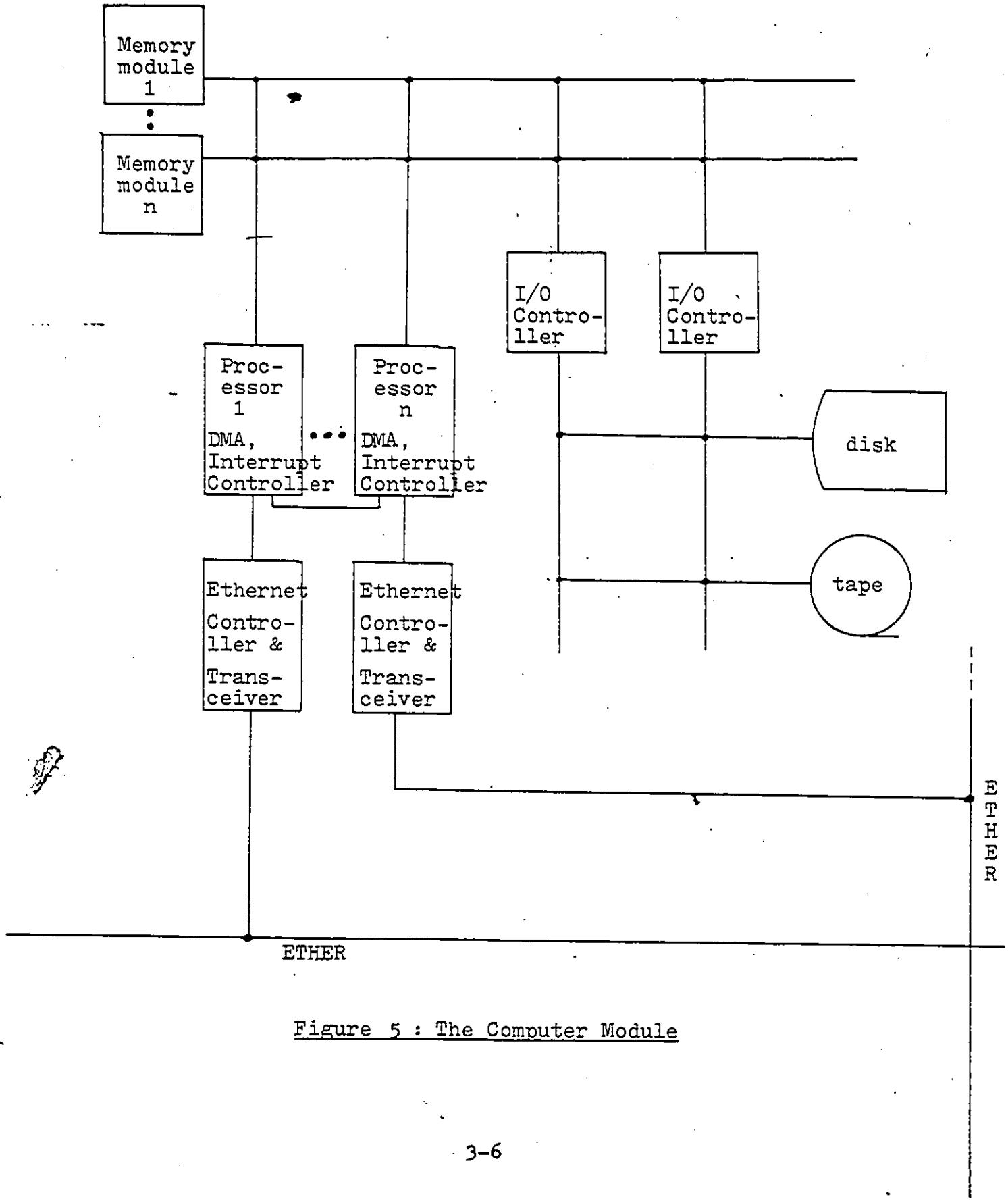


Figure 5 : The Computer Module

3.2 Distributed Ethernet.

Ethernet is a distributed packet switching system for local communication among computing stations [32]. The transmission media are coaxial cables. Stations connected to a coaxial cable communicate with each other by sending variable length digit data packets. A station's Ethernet interface connects bit-serially through a cable to a transceiver which in turn taps into the coaxial cable (the Ether). The communication mode is broadcasting, whereby all stations hear the same message. A station accepts a packet if the destination address on the packet is the same as the station's address.

Ethernet's communication protocol takes a strong influence from the Aloha Network. It uses a carrier sense multiple access with collision detected mode (CSMA-CD) for communication. Before attempting any transmission, an Ethernet controller senses for any transition on the Ether. The packet's bits are phase encoded which ensures that there is at least one transition on the Ether during each bit time. In analogy to sensing the presence of carrier in a multiple access path for radio communication, the mode of access is termed carrier sense multiple access. The CSMA protocol is further broken down into non-persistent and p-persistent protocol, which deals mainly with the retransmission algorithm. In Ethernet the retransmission

algorithm is called Binary Exponential Backoff, which is considered as a non-persistent CSMA. The retransmission intervals are multiples of a 'slot', the time for one end-to-end round trip delay. An Ethernet controller starts with a mean interval of one slot; each collision brings the delay interval of retransmission to a mean of twice that of the previous interval.

What distinguishes the Ethernet from the CSMA protocol is its provision of detecting interference. Interference is indicated when the transceiver notices a difference between the value of the bit it is attempting to send and the value of the bit it is receiving from the Ether. The advantages of this approach is to limit the interference period to a maximum of one round trip time. The packet will be able to be rescheduled for transmission as early as possible and the frequency of collision is used for the Binary Exponential Backoff algorithm.

In Ethernet, the acknowledgment of packets does not have priority over data packets. For reliability purposes, a modified Ethernet - Acknowledging Ethernet - gives priority to an acknowledge packet transmission over a data packet transmission. A simulation showed the Acknowledging Ethernet has shorter response times and increased effective transmission rates (34).

From another report on the performance of an Ethernet local network in real life, the statistics showed an Ethernet efficiency of 97.4% with 64 hosts and the packet length of 512 bytes [28]. However, the same report also showed an Ethernet efficiency of 54.2% with the same number of hosts and packet length of 4 bytes. Thus we can see Ethernet's performance is extremely good for file transfer applications and modest with interactive applications.

3.3 Merits of DCA.

There are many advantages of the Distributed Communication Architecture.

(1) Under Flynn's classification of computing systems, the DCA can be used to simulate the operations of a SISD, SIMD, MISD and MIMD machine, see Figure 6.

It is a SISD machine when the DCA architecture consists of only one computer module with only one processor.

It can simulate a SIMD machine when different transactions (data) are coming in through the x- axis cables and the first set of computer modules all execute the same sequence of operations.

It can simulate a MISD machine when data are broadcast on one cable and the computer modules execute different sequences of operations on the same data.

Finally, the computer modules themselves are MIMD machines and the DCA organization can therefore act as a MIMD machine. Although performance will be degraded when DCA is used to simulate operations of SIMD or MISD machine, DCA however, serves as a tool to study these structures.

(2) Reliability - the reliability of this system is a function of the number of shared components attached to the cables. DCA is reliable because the control function of the system is passive and fully distributed. Bus coupling is based on the use of transformers to DC-isolate all devices connected to the bus. Although distributed control increases the node

complexity, because of a redundant Ethernet interface in each computer module, the probability of losing the whole computer module is small.

Assuming now that p is the probability that a computer module fails. For a system of n by n structure, the maximum delay is 2(n-1) links delay, when $(n-1)^2$ computer modules are out of service, see Figure 7. This probability of 2(n-1) links delay is $(p^{(n-1)^2})(\ (1-p)^{2n-1})$. If $p = 10^{-3}$ and n = 10, the probability is about 10^{-243}.

(3) Expansion - there are many ways of expanding the system. It can be done by adding more processors to a computer module, by adding more computer modules to the Ether, or by increasing the bandwidth of the transmission medium. Fiber optic cables can be used to replace the coaxial cable, allowing a much higher bandwidth. An additional cable can link two or more computer modules, if the need arises in intermodule communication. The bottleneck of an application usually determines what option is required.

In order to give an idea of cost versus expansion, as viewed from the interconnection system point of view, the ratio of the number of links to the number of computer modules in a N by N configuration is tabulated in Table 3.

(4) Reconfiguration - system reconfigures to suit different applications and to share resources. As explained in (1), the architecture can be used to simulate four classes of machines. Since Ethernet is used to share resources in local computer networks, the DCA architecture inherits this property.

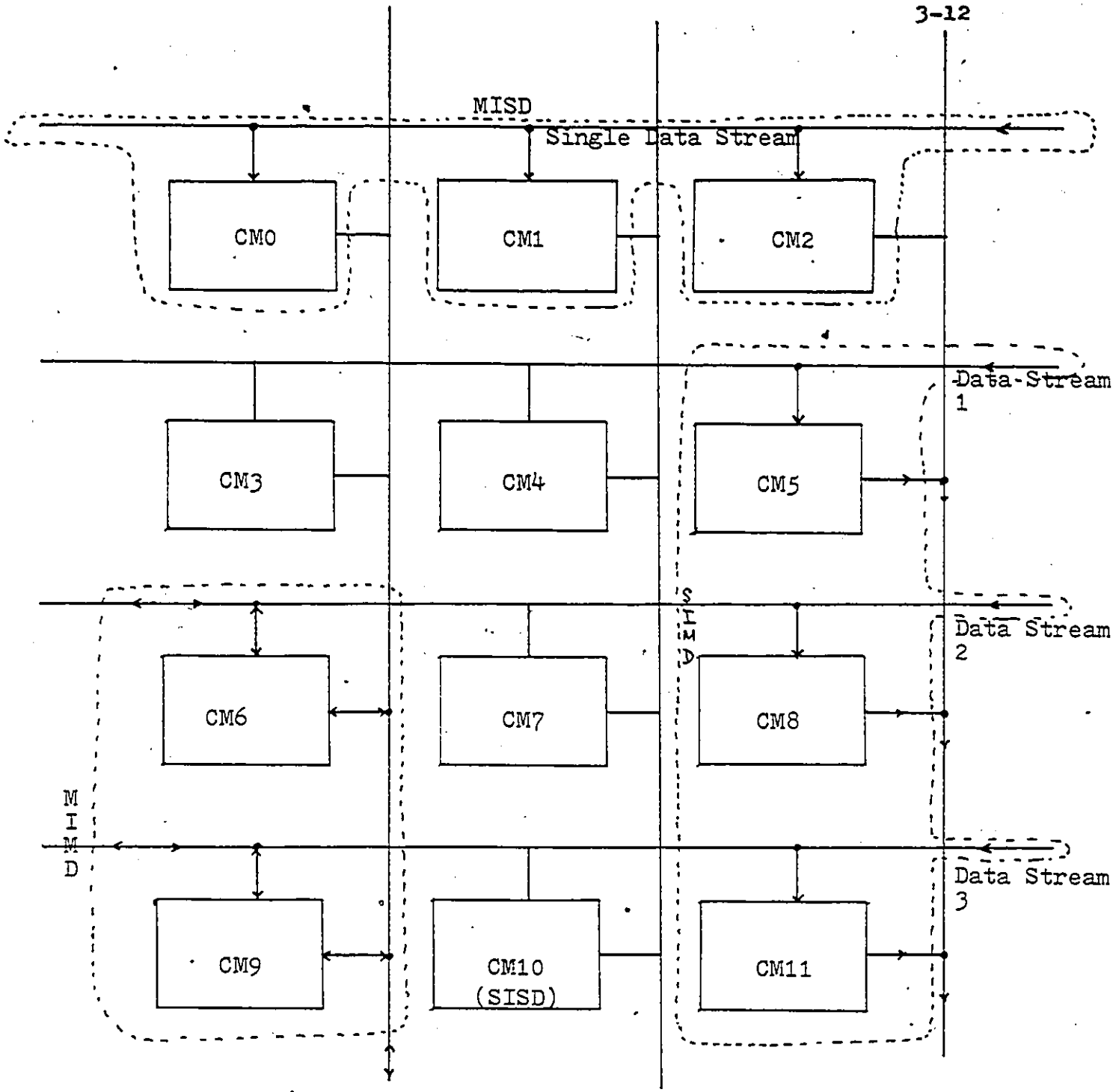
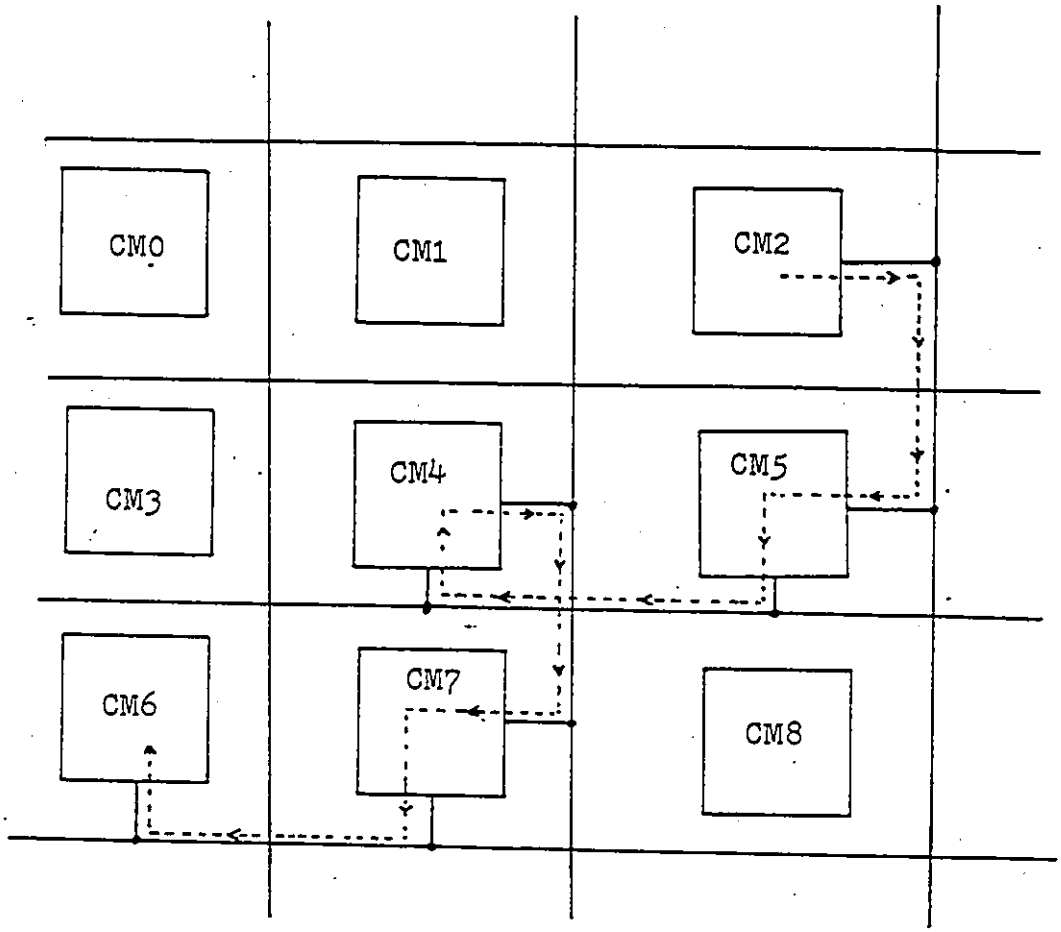


Figure 6 : Different configurations of Distributed **Communication Architecture.**



CM0,CM1,CM3 and CM8 are out of service.
One Ethernet Controller in CM2 and CM6 are out of service.

Figure 7: Maximum Number of Links Delay of DCA

Configuration	Computer Modules	Links	Ratio of Links to Computer Modules
2 x 2	4	4	1
3 x 3	9	6	0.67
4 x 4	16	8	0.5
5 x 5	25	10	0.4
6 x 6	36	12	0.33
7 x 7	49	14	0.29
8 x 8	64	16	0.25
9 x 9	81	18	0.22
10 x 10	100	20	0.2
100 x 100	10,000	200	0.02

Table 3: Expansion ratio of links and Computer Modules

3.4 Performance of Distributed Communication Architecture (DCA).

In this section, two examples are given to calculate the performance of the DCA structure. The first example calculates the performance of a computer module under ideal and non-ideal situations. The second example determines under what situation will the DCA structure switch from CPU-bound to I/O-bound in a transaction processing application.

3.4.1 A Picture Processing Application.

The problem is related to image processing. Given a picture composed of N by N elements and a window size of M by M elements, we want to find the highest value of the elements within the window, as the window moves from left to right and top to bottom of the picture. In other words, it is required to find the highest value of M^2 elements $(N-M+1)^2$ times.

To calculate the upper bound of performance on the DCA structure, we assume a configuration of the DCA system. In this example, we assume a memory organization of P-way interleaving and P 8086 microprocessors with a basic clock cycle of 125 nanoseconds. Let M=4, N=500 and P=4.

The elements in the picture are stored in different memory modules as shown in figure 8 . Each processor is able to access an element at each memory module simultaneously without any conflict. The access pattern corresponding to each time unit and each processor is shown in figure 9 . At each time interval, each processor accesses different memory modules. This configuration allows us to calculate the upper bound of performance on DCA.

Each processor inputs an element into the accumulator and compares the accumulator content with a designated register

content. If the accumulator content is larger than the register content, the accumulator content replaces the register content. A cycle is defined here to represent the time to find the highest value of the 16 elements. The cycle has its minimum value if the first element of the 16 elements is the highest value. Under this situation, the cycle takes $(21M^2)(125)$ nanoseconds. The minimum time to complete the whole job is

$$\frac{(N-M+1)^2 \times 21M^2 \times 125}{P \times 10^9} = 2.6 \text{ seconds}$$

Under a non-ideal situation, some of the processors will finish the cycle earlier than the others. Inevitably, there will be memory conflicts when more than one processors want to access the same memory module. To determine the expected number of processors to be active, Section 2.2 shows the formula to calculate the value, for the number of processors and memory modules in a system. Table 4 shows the job time under different configurations and bus situations. The corresponding graph is shown in Figure 10.

	Memory Module			
	1	2	3	4
Element	1	2	3	4
	5	6	7	8
	.....			
	501	502	503	504
	.....			
	249997	249998	249999	250000

Figure 8 : Storage Pattern of Picture Elements in 4-way interleaved memory system

	Processor				
	1	2	3	4	
Time Interval					
1	1	2	3	4	Picture Elements
2	2	3	4	5	
3	3	4	5	6	
4	4	5	6	7	
5	5	6	7	8	
6	6	7	8	9	
7	7	8	9	10	
8	8	9	10	11	

etc.

Figure 9: Access Pattern by Processor in different time intervals

Window Size	Number of Memory Modules, Number of Processors	Job Time under ideal situation	Job Time With Memory Interference
4 by 4	2,2	5.2 sec.	7.8 sec.
	4,4	2.6 "	4.6 "
	8,8	1.3 "	2.4 "
	16,16	0.65 "	1.3 "
	32,32	0.33 "	0.7 "
	64,64	0.16 "	0.3 "
8 by 8	2,2	20.4 sec.	30.6 sec.
	4,4	10.2 "	17.9 "
	8,8	5.1 "	9.6 "
	16,16	2.6 "	5.0 "
	32,32	1.3 "	2.6 "
	64,64	0.64 "	1.3 "

Table 4: Job time of picture processing application under ideal and memory conflict situation

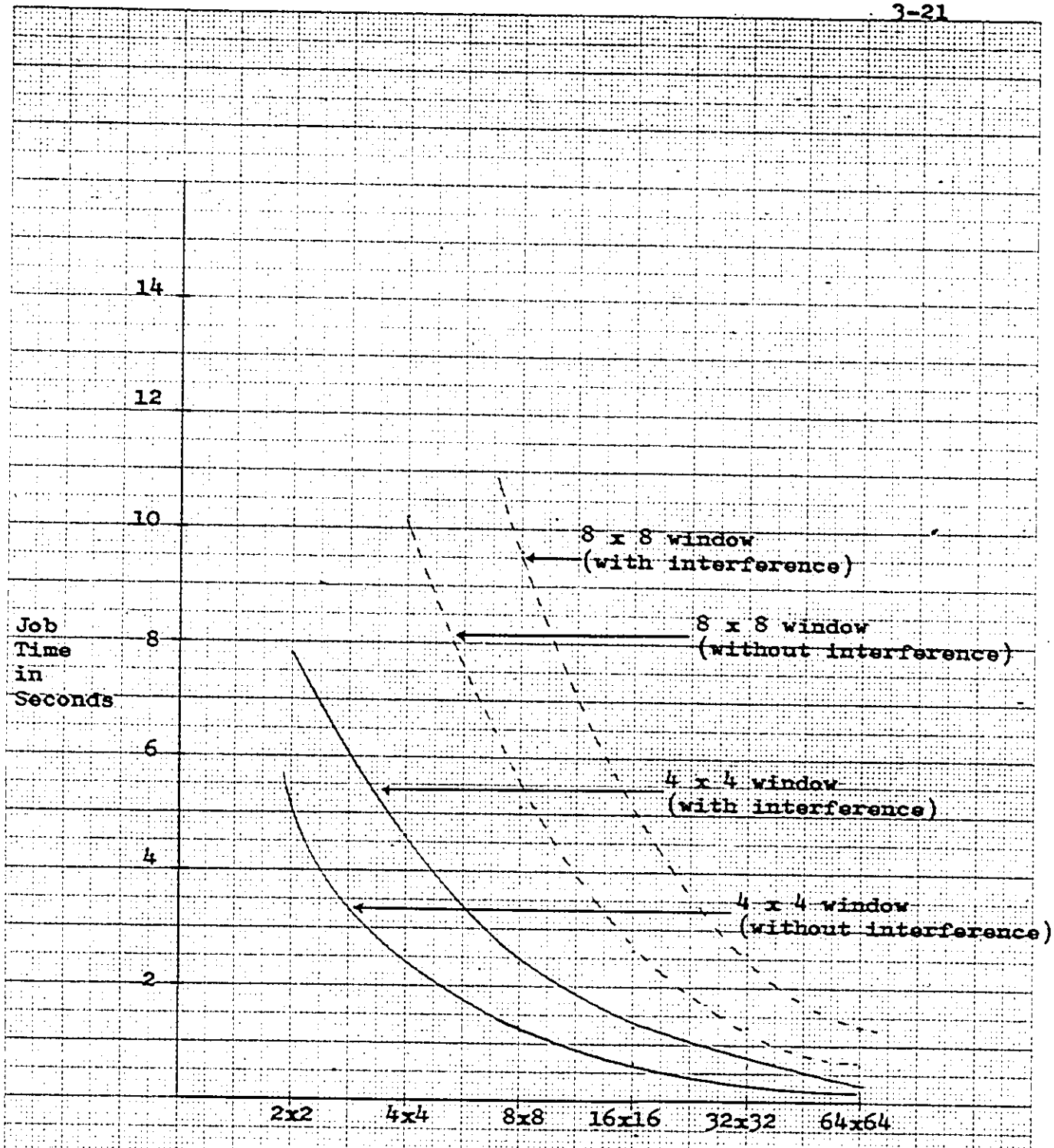


Figure 10: Picture Processing Job Time under different configurations and with or without memory interference

3.4.2 A Transaction Processing Application.

A process in a station (i.e. a computer module) receives a transaction via a terminal, processes the transaction and sends a message to another station. The process waits for the next transaction before repeating the same cycle again.

Let M be the mean number of transactions transmitted per second over a link (i.e. the Ether). Then the mean transmission time of a transaction is 1/M second.

Let T be the mean number of transactions processed in one second by a station. Thus, the mean process time is 1/T second.

The traffic intensity, T(I), is defined as M/T.

Let p1 be the probability that the message is sent to a station on the same link, and p2 be the probability that the message has to transverse through two links. The mean transmission time is

$$p1\frac{1}{M} + p2\frac{2}{M} = (p1 + 2p2)\frac{1}{M}$$

Assuming there are N by N stations in DCA, as shown in figure 11, thus

$$p1 = \frac{2(N-1)}{N^2-1} \quad ; \quad p2 = \frac{(N-1)^2}{N^2-1}$$

The mean transmission time is

$$\left[\frac{2(N-1)}{N^2-1} + \frac{2(N-1)^2}{N^2-1}\right]\cdot\frac{1}{M} = \frac{2N}{N+1}\cdot\frac{1}{M}$$

The average number of links to transverse in a N by N configuration is equal to 2N/(N+1).

Since the mean number of transactions on a link is M transactions per second and there are 2N links, the maximum number of messages in DCA is 2NM per second. The mean number of messages per second on a link is

$$2NM\Big/\left(\frac{2N}{N+1}\right) = M(N+1)$$

In other words, M(N+1) is the maximum number of processes that can be generated per second in a 'Bus-Bound' situation. The total processing time, PB, in the DCA system per second is

$$M(N+1)\cdot\frac{1}{T}$$

In a 'CPU-Bound' situation, the fraction of process execution time per second is

$$\frac{1/T}{1/T + \left(\frac{2N}{N+1}\cdot\frac{1}{M}\right)} = Y$$

The total processing time, PC, in the DCA system per second is

$$N^2Y$$

When PC = PB, the system switches from CPU-bound to I/O-bound. This simplifies to

$$N = \frac{(2T+M) + \sqrt{(2T+M)^2 + 4TM}}{2T}$$

If $(2T+M)^2 >> 4TM$ or $T >> M$, then $N = 2 + M/T = 2$. Thus a 2 by 2 configuration yields a good balance between processing and I/O operation when $T >> M$.

Table 5 shows some combinations of T and M values and their resulting values of N. As the processing time becomes the dominant factor in a transaction cycle, it becomes feasible to add more computer modules to increase the overall system throughput. Given the message rate and the processing rate of a transaction, the number of computer modules in DCA can be estimated to yield a balanced system.

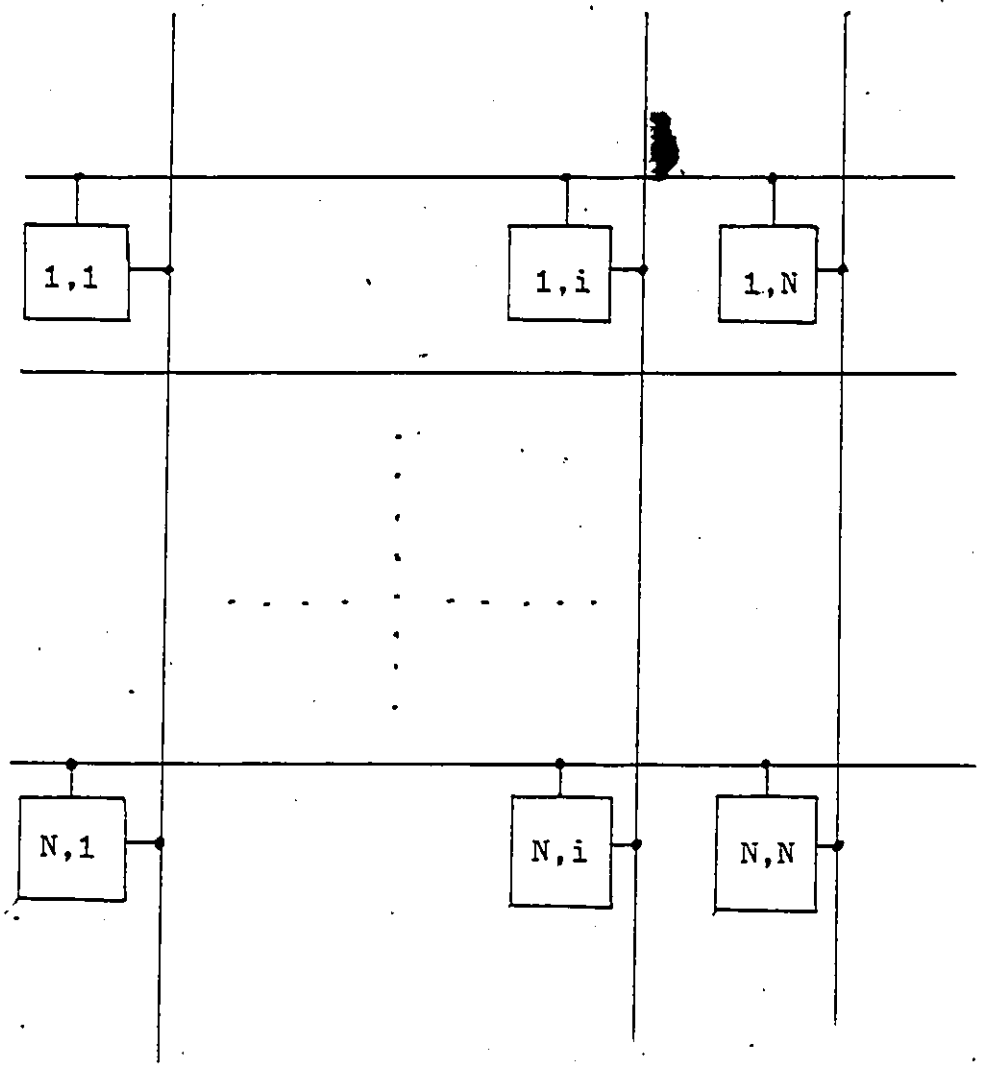


Figure 11: An N by N configuration of DCA in transaction processing

T	M	N
100	10	2
100	50	3
100	100	4
100	200	5
100	400	7
100	1000	13

T: Mean number of transactions processed per second.

M: Mean number of transactions transmitted per second

N: Number of Computer Modules in a N by N configuration

Table 5: Transaction Processing Configuration Calculation.

3.5 Conclusion.

DCA, a distributed communication architecture, is presented in this chapter. The organization can be used to simulate the operations of a SISD, SIMD, MISD or MIMD structure. DCA can also be used as a local area network. The Computer Module, the basic unit of DCA, is a multiprocessor with a crossbar switch interconnection system. An Ethernet-based network connects the computer modules along a multiple access transmission medium. The architecture is suitable for expansion, reconfiguration and reliability.

CHAPTER 4 - INTERPROCESS COMMUNICATION IN DCA

This chapter examines the interprocess communication in a distributed system. The issues of distributed operating systems and distributed data bases are not discussed, due to their diverse scope.

4.0 Interprocess Communication in DCA.

One of the major issues in a distributed system is the communication requirements between processes [13]. A process is a virtual or logical processor. A job or a task requires many execution cycles of the processes. Processes do not always act independently; they communicate with each other to coordinate activities. One of the requirements of interprocess communication is unambiguous synchronization; the ability to access shared resources in an orderly manner. Another requirement is the maintainence of standard or uniformity in interprocess communication techniques. Since there may be heterogeneous hosts in a distributed architecture, each having dissimilar process level protocols, it is desirable to avoid any kind of transformation between them. Transformation requires processing time and space in terms of memory, and is best avoided. By providing resources transparency to all users, it also allows portability and uniform access to resources that are local or remote. Finally, the performance objectives should also be considered. They are throughput, delay, cost effectiveness, and reliability.

In the Distributed Communication Architecture (DCA), the

interprocess communication can be looked upon from two different aspects. The first aspect concerns the communication within a Computer Module (CM), or the intra-module interprocess communication. The second aspect concerns the communication among Computer Modules, or the inter-module interprocess communication. The rest of the chapter is devoted to these two aspects, and also the process control structure required to implement the interprocess communication scheme.

4.1 DCA Intra-module Interprocess Communication.

4.1.1 Implicit and Explicit Mechanisms.

Intra-module interprocess communication mechanism takes many forms. The interprocess communication mechanisms for common memory systems have had a long history - from Dijkstra's semaphores [14][15], Hansen's conditional critical regions [16], Hoare's monitors [17] to Campbell's and Habermann's path expressions [18][19]. All of these mechanisms are best suited to solve synchronization problems. They are implicit in nature because they share a common object and have to follow a certain access rule to keep the object in a consistent state [20]. Their access operations are known as procedures.

The message passing system [21] is an explicit synchronization system in which a process has to wait until another process has completed an action, which results sending a message to 'wake up' the calling process. It is not necessary, however, that the calling process goes to 'sleep' after it sends a message to a called process requesting for service, if the calling process can continue without the result from the completion of that service. The calling process can proceed asynchronously along with the called process until it requires a feedback from the called process before it can proceed further.

4.1.2 Criteria to choose an interprocess communication mechanism.

In order to determine the type of interprocess communication mechanism we need, it is necessary to look again at the requirements. The major requirement in the present case is a uniform implementation technique for a distributed system. This technique should be able to survive, if the system is separated into different computing units (e.g. from an 8-processor system with common memory to two 4-processor systems without common memory), and withstand integration of two or more units into one. Based on this stringent requirement, the message-oriented system is chosen because it can be used in common memory systems and distributed systems.

4.1.3 Characteristics of a Message-Oriented System.

From a good design point of view, there are several outstanding features in a message-oriented system (22).

- Resources of the same type are associated with their own processes which act like resource managers. If a process requests for a resource, it sends a request in the form of a message to the process associated with the resource. In this way, requests are received and processed serially. This serial characteristic avoids any synchronization problem. Unlike

the other implicit synchronization techniques which allow many processes to carry out the synchronizatioh procedure themselves, the process-oriented resource management restricts the synchronization task to itself.

- Process preemption, the blocking of a process, has less delay effect with accessing common resources in a message-oriented system. Since there is a single process in charge of a resource, preemption has delay effect only on that particular process. The implicit synchronization techniques allow many processes to enter the allocation and deallocation regions, and thus process preemption of any of these processes may have a delay effect on each other.

- Communication paths are established between pairs of processes. The path ends specify the station number, process number and port number. A port is a queue used to hold messages belonging to a certain class.

- Connections between processes are relatively stable. A 'father' process can be used to receive and dispatch all messages, while at the same time creating 'son' processes to handle the received messages.

4.1.4 Message-oriented Operations.

There are four primitive operations in a message-oriented system. They are connect, disconnect, send and receive[23][24]. The

formats of the four operations are:

```
connect (sender, receiver)
disconnect(sender, receiver)
send (receiver, message)
receive (sender, message)
```

The connect operation is used to establish a logical connection between two processes. The operating system tries to locate the destination process, initiates the connection protocol and waits for a confirm message from the destination process before it sets a condition return code in the source's process control block. The sender is blocked until the operating system is ready to return the condition code. In addition, connection information is updated into the process control block of the sender.

The disconnect operation does the reverse of a connect operation, with the difference of initiating a disconnect protocol.

The send operation copies the message into a buffer, finds the receiver location from the process control block and passes this information to the communication processor. A broadcast operation can be carried out by decoding the receiver address in which a common or global address can be used. The buffer is attached to the sender's process control block for a future receive operation. In this way, a send-receive pair of operations guarantees a buffer space is

available.

The receive operation essentially waits for a message. Whether the process wishes to be blocked or not depends on the operating system implementation. If a process cannot proceed further without the expected message, then it should be blocked to allow other ready processes to gain control of the processor. In the case where a process can proceed further, it makes no sense to spend time on switching to another process. A process can inform the operating system on blocked receive by issuing a system call 'wait' after the receive command. When a message arrives, it is copied from the reserved buffer to the process message space. The buffer is put back into the buffer pool. The process is unblocked and put into the ready list.

In addition, the operating system must be able to handle messages to or from a process which has been destroyed or removed. Those messages sent to a nonexisting process must be returned with status information. Other messages can be left in the queues of the other processes.

4.1.5 Analysis of a Message-oriented System.

The concentration of requests and service on a single resource process makes the analysis much easier. In fact, using the classical queueing system M/M/1, the average queue size and the average time delay can be found. The M/M/1

system assumes a Poisson arrival distribution, exponential service-time distribution and a single server (25).

Let the mean arrival rate be A(mean) requests/second, and the maximum service rate be A(max) requests/second. Thus the traffic intensity ,T, is A(mean)/A(max).

The average queue size is then given by E(n)=T/(1-T).

The average time delay is given by E(t)=1/(A(max)-A(mean)).

4.2 DCA Inter-Module Interprocess Communication.

The packet protocol, EFTP, used in Ethernet has been very useful for file transfer from station to station. It is not designed for process to process communication within a network. There are no mechanisms to relate processes and allow transparency in interprocess communication. There is no control function imbedded in the packet protocol to allow flow control or other functions such as remote program calling, contract broadcasting and coordination of resources. In order to take a broader look at interprocess communication, the following sections are dedicated to the discussion of the DCA protocol.

4.2.1 DCA packet protocol.

The DCA packet layout is shown in figure 12. A start bit called SYNC, as used in Ethernet, precedes the packet. This start bit is generated by the transmitting interface. The receiving interface uses the SYNC bit to detect the start of a packet and acquires bit phase. The rest of the packet is made up of multiple bytes. The first byte is the 'field length', followed by the destination address, the source address, a sequence number field, the control field, data information and a 16-bit cyclic redundancy checksum.

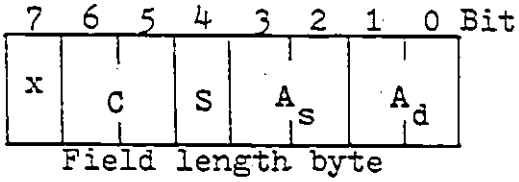
The 'field length' byte is used to indicate the length of the source address, destination address, sequence number

field and the control field. The breakdown of the field within the 'field length' byte is shown in figure 13.

SYNC	FIELD LENGTH	DESTINATION ADDRESS	SOURCE ADDRESS	CONTROL FIELD	DATA FIELD	16-bit CRC

Figure 12

DCA Packet Format



A_s,A_d = 00 to 11, address field length is 1 byte to 4 bytes respectively, for source (A_s) or destination (A_d);
Station address (1) byte,
Process address (2) bytes, and
Port address (1) byte.

S= 0 to 1, sequence number field length is 1 byte to 2 bytes respectively.
Ns or Nr = 0 to 15 OR 0 to 255 respectively.

C = 00 to 11, control field length is 1 byte to 4 bytes respectively.

x= not used.

Figure 13

Field Length Byte

4.2.1.1 Flag.

A flag is used to denote the beginning of a packet. In IBM's Synchronous Data Link Control (SDLC) protocol, the '01111110' bit combination is used as a flag or what is also referred as a 'sync' character. In the DCA network, it follows the Ethernet's convention with a 'sync' bit to indicate a transition from idle to transmission of packet. Bit or byte stuffing is therefore deemed unnecessary.

4.2.1.2 Destination and Source Address.

An address is used to identify a unique entity from another. A station has an address which is distinct from other stations, and within stations, other entities like processes have their own addresses. In fact a process contain several addresses with each address dedicated to communication with another process[27]. The address can be dynamically created, transferred or destroyed; thus providing security in the system.

In DCA, the address field can be from 1 to 4 bytes long. The first byte is the Computer Module's (CM) address, the next two bytes is the process number within the CM, and the last byte is the port number within a process. The port number consists of two fields, a 7-bit field designating the port address and the least significant bit indicating a send port if it's a zero and a receive port if it's a one. Thus a

process consists of 128 ports for output and input. The address field format is shown in figure 14 .

8	16	7	1	bits
Computer Module Address	Process Address	Port Address	S/R	

S/R= Send Port or Receive Port

Figure 14

Address Format

for source or destination

4.2.1.3 Sequence number field.

A sequence number can be used for error control. In SDLC, a sequence number is made up of three bits [26]. Stations transmitting sequenced information packets request confirmation by sending a 'send sequence count', Ns; the other station confirms by sending a 'receive sequence count', Nr. Both stations keep a pair of counts, Ns and Nr, to designate the next packet number to send and the next packet number to receive respectively. The counting capacity for Ns and Nr is 8; allowing up to 7 frames to be sent before a receiver reports its Nr count to the transmitter.

To determine the maximum value of a sequence number, there are two factors to be considered. One is the maximum data rate (number of packets per second) in the CM, and the other is the maximum packet life time in the DCA network. For example, if the maximum data rate in the CM is 1000 packets per second, and the maximum packet life time in DCA is 250 milliseconds; then the maximum sequence number is 1000 x 0.25 = 250. Any value greater than this maximum sequence number is guaranteed (in theory) to safeguard any duplicate sequence number.

In the DCA packet protocol, an option is provided for a user to use a maximum sequence number of 15 or 255. The field length byte contains a bit to designate what option is

used. Because the transmitting medium and the maximum data rate can be changed dynamically, the option is used to allow flexible implementation for DCA.

4.2.1.4 Control field.

The control field is used to indicate the message intention or message types. Depending on this information, a process can coordinate with or activate other processes. There are various types of control information for different applications, so in this section the description is restricted to general usage. Figure 15 summarizes the control functions. Individual functions are explained below.

(1) Positive Acknowledgment - reception of message is error free.

(2) Negative Acknowledgment - error in received message or intention reject

(3) Positive acknowledgment with buffer requirement - in addition to (1), it informs the receiver how many more messages it can take.

(4) Positive acknowledgment , stop transmission until further notice - the received message is error free but no more messages should be sent until further notice.

(5) Restart transmission - receiver can resume receive operations.

(6) Locate program - locate a remote program or process.

(7) Remote call - submit job to a remote program.

(8) Remote return - remote program indicates work is finished.

(9) Request resource - request for a resource

(10) Allocate resource - request for reservation of a resource.

(11) Deallocate resource - indication to return the resource.

(12) Confirm resource - resource is available .

(13) Data transfer - transfer parameter or data with the format of data following the control field.

(14) File transfer - transfer large amount of data, with the format of data and the class of device in which the file is saved on following the control field.

(15) Reset frame sequence number - reset the next expected message sequence number.

(16) Connect - request for a virtual connection.

(17) Disconnect - request for a disconnection.

(18) Pace - receiver cannot keep up with the transmitter's speed, do not send any more messages until further notice.

(19) Request for bidder - given a job description in the message, it requests for a station which has the resources to submit a bid.

(20) Submit bid - station contains all the resources required for this job, the percentage workload and downtime on this station is indicated in the message.

(21) Award bid - the station wins the contract.

(22) Reject - request retransmission of Nr and following frames.

(23) Selective reject - request retransmission of Nr only.

(24) Reconnect - previous connection was broken, initiates connection again.

Control Field

Positive Acknowledgment
Negative Acknowledgment
Positive Acknowledgment with buffer requirement
Positive Acknowledgment, stop transmission until further notice
Restart Transmission

Locate Program
Remote Call
Remote Return
Request Resource
Allocate Resource
Deallocate Resource
Confirm Resource

Data Transfer
File Transfer

Reset Frame Sequence Number
Pace

Connect
Disconnect
Reconnect

Request for bidder
Award Bid
Submit Bid

Reject
Selective Reject

Figure 15: Summary of Control Field

4.2.1.5 Information field.

The information field contains data that is transported from one port to another in the DCA system. The data may be physically transmitted from one CM to another via the Ether; or internally by data buffer descriptor. A descriptor contains a pointer to the starting address of a buffer, the number of items of data in the buffer and the length of an item.

4.2.1.6 Error check sequence field.

The error check sequence field contains a 16-bit cyclic redundancy checking (CRC) code. The CRC code is the remainder generated by dividing the bits stream after the SYNC bit by a polynomial. The international standard for the polynomial, CRC-CCITT, is $X^{16}+X^{12}+X^{5}+1$. By recomputing the CRC upon receipt of the packet, the receiver can verify the content of the packet.

4.2.1.7 Packet size.

An optimal packet size is determined by a tradeoff between the requirements of delay and throughput. A short packet size has the effect of lower delay and lower throughput, and more software overhead on the system. A long packet size has the effect of higher throughput and higher delay, and a higher error probability on the system. A preliminary report on Ethernet (28) shows a mean packet length of

about 122 bytes; with an efficiency of more than 90% with 64 hosts.

In DCA, we choose a packet size of 1K bits (128 bytes) as the unit of communication. In the future, when the transmission medium quality improves, for example with lower error rates and higher bandwidth, the influence of packet size will decrease.

4.2.1.8 Data transparency.

Data transparency implies distinguishing data bits from control bits such as the start-of-message character. In the DCA system, by using a SYNC bit to indicate the start of a message, data transparency is inherited in the protocol. Unlike IBM's Binary Synchronous Control and SDLC disciplines, byte stuffing or bit stuffing is not required in DCA.

4.2.1.9 Connection establishment and clearing.

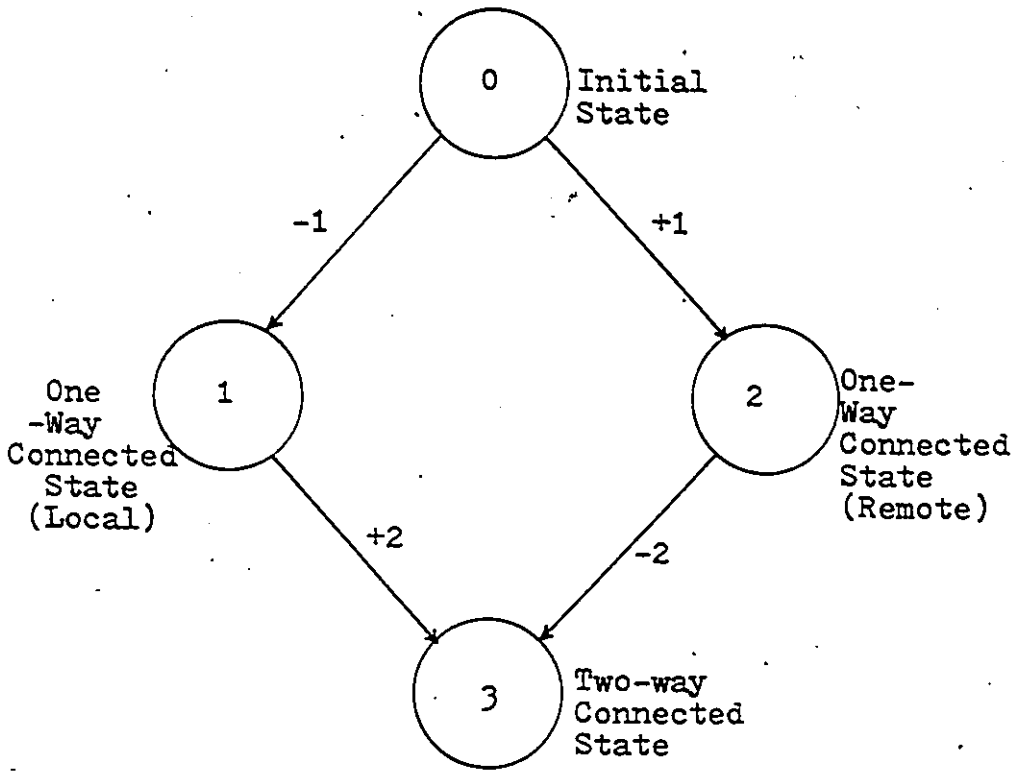
Before any exchange of data between two processes, it is necessary to determine whether the called process is operating or willing to establish a connection. Similarly at the end of a conversation between two processes, one process must initiate a disconnection. In the case of system recovery after failure, a reconnection between a previous existing conversation is needed.

Under these three situations, a process must (1) identify itself, (2) determine what is the sequence number, and (3) exchange any data that are necessary, e.g. process identification and port number, reasons for disconnection or accounting information etc..

A connection request is confirmed by the called process by also sending a connection request to the calling process. The information will be the same but the value may be different; in both cases a receiving port number and a send sequence number are passed to the other process.

The control field is used to indicate a connect/disconnect function. By decoding the control field, the connection establishment and clearing function can be realized.

The state diagram for a connection is shown in figure 16



1 Connect Request
2 Confirm Message
- Send
+ Receive

Figure 16: Connect State Diagram

4.2.1.10 Error control.

Error control consists of detecting transmission errors, missing or out of sequence packets and duplicate packets. It is enforced by parity checking and cylic redundancy checksum to detect transmission errors. The sequence number imbedded in a packet is used to uniquely identify a packet. A timeout mechanism is imposed on a packet's lifetime in the network. Retransmission takes place if a negative acknowledgment is received or there is an absence of acknowledgment within a time frame.

In the DCA network, under ideal situations, it is necessary to make at most two hops to reach a called party. Instead of giving a positive acknowledgment to the calling party right after the go-between party has received the transit packet, the go-between party seizes the Ether and forwards the transit packet to the called party. If there is no error when the called party receives the packet, then a positive acknowledgment is passed to the go-between party. At that time the go-between party then sends a positive acknowledgment message to the calling party. Within a Computer Module, an acknowledgment message has always the top priority to transmit.

Retransmission will also take place if the sequence number in the acknowledgment message is not the expected

value, or there is a timeout on the packet to indicate an absence of acknowledgment within the packet's lifetime in the network. In either case, a selective or reject retransmit scheme will cause the packet(s) to be sent again. The sender should keep a count of retransmission to indicate the status of a line and the receive party. The count can be used to determine a complete failure (permanent or intermittent) on the transmission medium and the receive party.

4.2.1.11 Flow control.

In order to balance the rate of sending and receiving data in a station, flow control is used to regulate the amount of data sending or receiving to or from a station. This scheme is called the 'window technique' in which the limit is referred to as a 'window'. It indicates the number of outstanding packets or messages that can be output without receiving a positive acknowledgment. For a receiver, the acknowledgment message contains the sequence number of the next expected frame and also a window to indicate how many more frames it can accept. This can dynamically change the buffer allocation requirement and prevent overflow in a receiving station. Alternatively, a receiving station can explicitly send a message to the transmitter to indicate a halt transmission. At a later time when the receiver can receive more messages from that station again, it can explicitly send a restart message to the transmitter. In all cases, the flow control requirement is to regulate data rate

to/from a station so all stations within a network can work harmoniously.

In DCA, if the sender and receiver are on the same Ether, the acknowledgment comes back right away and the window size for sending is set to one. The acknowledgment can contain a number to indicate a window for receiver. Instead of adopting a discard and wait for retransmission policy, it is more time saving to indicate the buffer or load situation at the receiver to the sender.

When the sender and the receiver are not on the same Ether, a third party is involved. Since there are 2 hops involved, the window size is initially set to 2 [29]. It is further observed that even with different arrival rates, the 'optimal' window sizes remain close to those for symmetric loading [30]. A higher network power can result with a smaller window size.

4.2.1.12 Buffer allocation.

To determine an optimal buffer space in a station, the probability that a packet will be discarded because of no buffer space is used to evaluate the allocation policy.

Let RHO be the ratio of the mean arrival rate to the mean service rate, and k the buffer size. Using a single server system and assuming exponentially distributed interarrival

and service times, the probability that k buffers are full is [31]

$$\frac{(1-RHO)(RHO)^k}{1-(RHO)^{k+1}}$$

For RHO >> 1, the probability approaches one meaning most arriving packets will be discarded. The buffer size has no influence to the system as long as RHO >> 1. For RHO << 1, the probability approaches zero meaning the buffer size can be very small because arriving packets are processed very quickly.

For RHO = 1, meaning the arrival rate is the same as the service rate, the probability is 1/(k+1).

In reality, application software normally allocates a pool of buffers for all the input/output ports. This can average out the usage of buffer space and allow some flexibility.

4.2.1.13 Waiting time and efficiency.

A simple model is used to calculate the mean number of slots of waiting in a contention interval and the fraction of time the Ether is carrying good packets [32]. Assuming there are Q stations continuously queued to transmit a packet and the probability that a queued station attempts to transmit in the current slot is 1/Q, then the delay probability is 1-(1/Q).

The probability that only one station attempts a transmission in a slot is $A=Q*(1/Q)*((1-(1/Q))^{Q-1})$ or $(1-(1/Q))^{Q-1}$.

The mean number of slots of waiting, $W = \sum_{i=0}^{\infty} i*A*(1-A)^i$ or $(1-A)/A$.

Let P be the number of bits in a packet, C be the channel capacity and T be the time in seconds of a slot, the number of seconds it takes to detect a collision after starting a transmission. The efficiency E is therefore
$(P/C)/((P/C)+(W*T))$

4.3 Process structure.

A process control block (PCB) is a process state descriptor. It contains information relating to the status of a process, the ongoing interactions with other processes and the resources it holds. At any time during the life cycle of a process in the system, the process control block must reflect an unambiguous current state of the process.

Process control blocks are linked together under different process states. A process can migrate from one state to another as shown in figure 17. Therefore a process control block is linked to the appropriate process state to reflect the current status of the process. Within the process control block, it contains queue headers pointing to different data structures. A data structure is used to describe a transaction or an object (e.g. resources). Any other pertinent information related to a process is also saved in the process control block. To give a better visual description, a process control block and its related data structures are shown in figure 18.

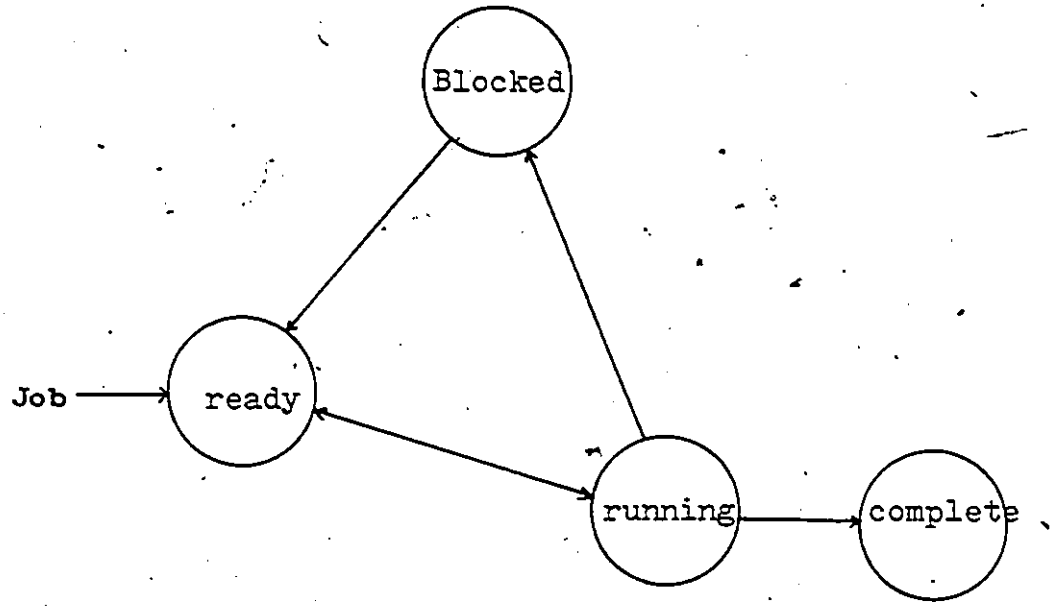


Figure 17

Process State Diagram

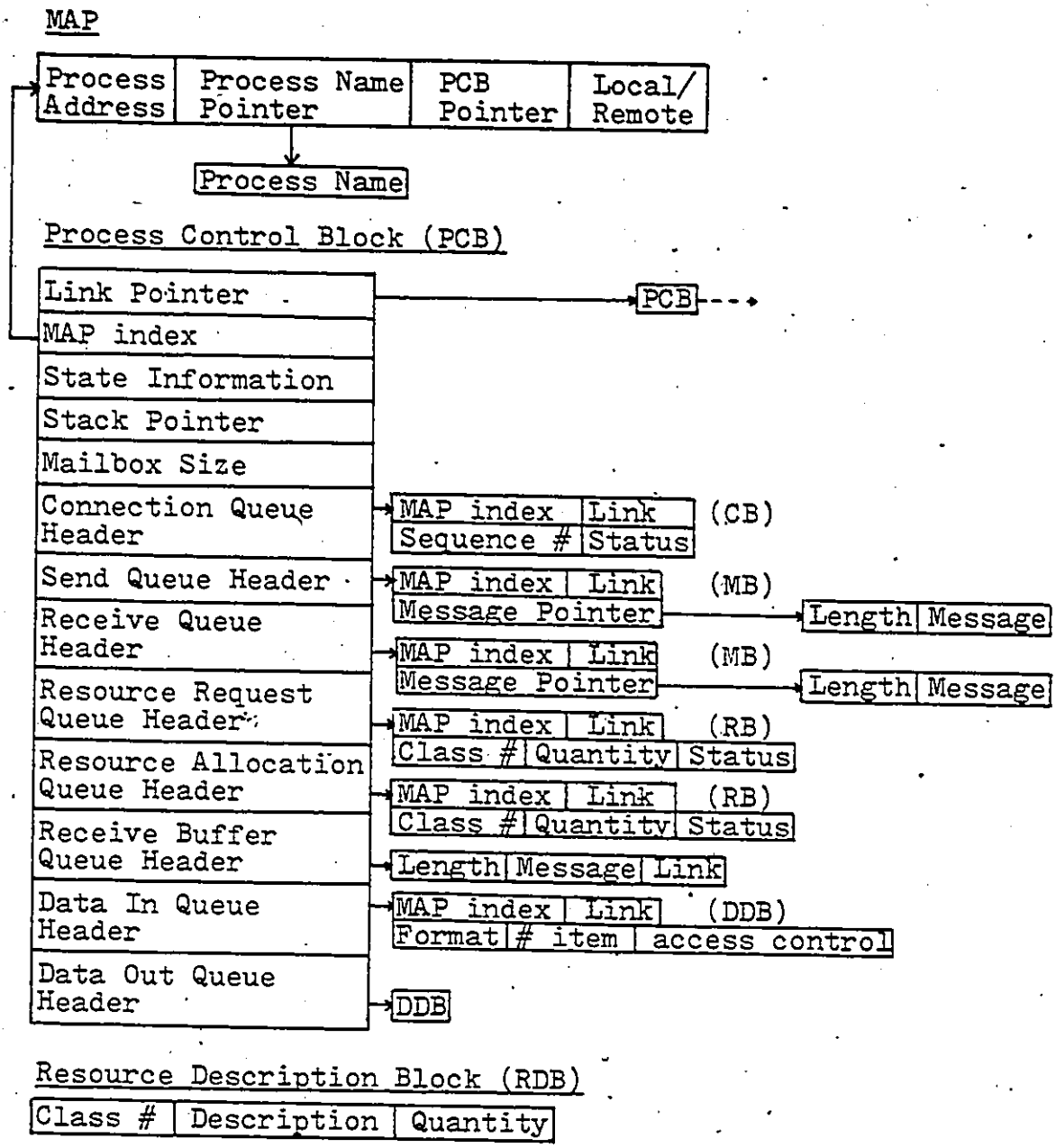


Figure 18.
Process Control Block & Data Structure

4.3.1 Data structures associated with a process control block.

The MAP data block is used to transform process names to process addresses. There is a bit to indicate whether the process is local or remote. This bit is set at process creation time or when a connection is made with another remote process. In addition, the address of the process control block is stored.

The resource description block (RDB) contains the class number of a resource and its description.. The class number is used to designate the types of resources (e.g., a 300 lines/minute line printer, a double density floppy disk etc.), and the description contains information such as the logical device name and address, quantity and status of the resource.

The connection block (CB) contains a MAP index of the destination process, the sequence numbers and status information. The status information contains bits setting to reflect the following:

- a process initiates a connection to a destination process
- a confirm on connection from the destination process
- receiver requests halt transmission until further notice

A message block (MB) indicates the sender or receiver MAP

index and a pointer to the message buffer.

A resource block (RB) shows the quantity of a resource class which is being held by a process or allocated to a process. The status bit shows the allocation state.

The data description block (DDB) contains the format of the input or output data, the number of items and access control information (e.g. read only, read/write etc.) related to the data.

The PCB also contains the process state information, a pointer to its own stack, and the size of its mailbox.

4.3.2. Example of a process-to-process transaction.

Process A contains hardware resources of disk and tape units. At a certain time, process A is switched to process transactions which are output to another process B after processing. The disk and tape units are not going to be used.

At first, process A establishes a virtual connection with process B. A connection block is linked to the connection queue. When process B acknowledges the connection, the status bit is set in the connection block to indicate a confirmed connection. Process A now has the priviledge to talk to process B, and vice versa.

When process A receives a broadcast message from process C with a control field indicating a 'request resource' type,

it checks the class of resources it holds. Assuming the request is for disk and tape units, process A constructs a resource block and attaches to its resource allocation queue, with the allocation status bit set to 'open'. It sends a 'confirm resource' message to process C. When process C responds by sending an 'allocate resource' message, process A sets the allocation status bit to 'close' and returns a 'deallocate resource' message to process C.

Now transactions are coming into the receive queue (assuming previous connections with other processes). Process A acts upon the transactions and moves the message block from the receive queue to the send queue, with the MAP index now pointing to process B. The processed transactions are sent to process B.

When process C finishes with the disk and tape units, it sends a 'deallocate resource' message to process A. Process A checks its resource allocation queue , accepts the message and dequeues the resource block.

Process A continues to interact with process B until it has finished all the transactions. At that time process A issues a 'disconnect' message to process B which confirms by returning a 'disconnect' message. Process A issues a system call to the operating system kernel which deallocates all the resources held by process A and removes the process. A

flowchart is shown in figure 19 for this process to process transaction example.

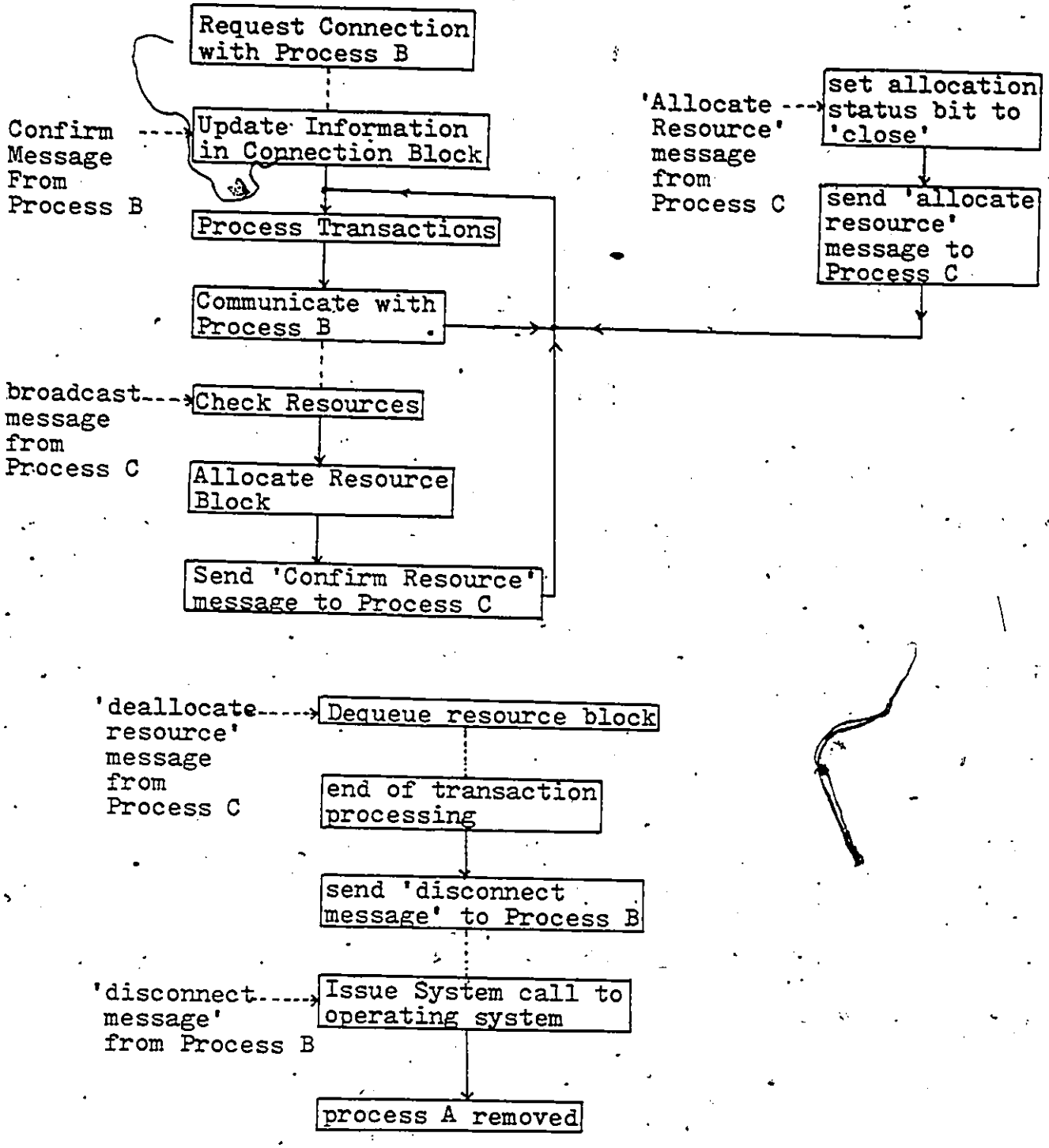


Figure 19: Diagram for process-to-process transaction example.

4.4 Summary.

This chapter has covered the requirements of an interprocess communication mechanism, the choice to determine what is the best for the Distributed Computer Architecture, the characteristics of the mechanism, the operations associated with it, the design and issues with the protocol, the performance calculation and the process control structure.

Although this chapter will not be able to cover every aspect of interprocess communication, it gives an architecture which can be refined in subsequent detail designs to arrive at a working model. There are special groups working on standard protocols for open system architecture and it will arouse further interest in this area.

CHAPTER 5 - CONCLUSION

5.0 Conclusion.

This thesis focuses on the design and analysis of a distributed communication architecture, DCA, and its interprocess communication mechanism.

The DCA structure is made up of computer modules which are interconnected together by a communication medium (e.g., coaxial cables) to form a unique architecture. The communication discipline is based on Ethernet. This distributed architecture can be used to simulate a SIMD structure (e.g., an array processor), a MISD structure (e.g., a pipelined processor) or a MIMD structure (e.g., multiple computer system). The DCA system can also be partitioned into units with each unit operating as any structure (SIMD, MISD or MIMD).

A computer module contains a crossbar switch interconnection system connecting processors with memory modules. From the analysis of the crossbar switch, as the number of processors and memory modules increases together, the average number of active processors at any interval is about 50%. This indicates that, by adding more processors and memory modules to the computer module, it still maintains half of its total throughput.

The DCA system is reliable because of its redundant paths interconnecting processors and memory modules within a computer module, and between computer modules. This interconnection system also allows expansion at moderate rate and reconfiguration for special applications.

An interprocess communication mechanism is provided to allow processes to coordinate activities and share common resources. A uniform approach is suggested to eliminate any transformation. The interprocess communication protocol contains facilities to call remote programs and solicit for work or resources.

REFERENCES

(1) P.H.Enslow,"What is a "Distributed" Data Processing System ?", IEEE Tutorial: Distributed Processor Communication Architecture, October 1979, pp.3-11.

(2) K.J.Thurber,"Computer Communication Techniques", IEEE Tutorial: Distributed Processor Communication Architecture, October 1979, pp.12-17.

(3) G.A.Anderson and E.D.Jensen,"Computer Interconnection Structures: Taxonomy, Characteristics, and Examples", IEEE Tutorial: Distributed Processor Communication Architecture, October 1979, pp.24-40.

(4) E.D.Jensen,K.J.Thurber, and G.M.Schneider,"A review of Systematic Methods in Distributed Processor Interconnection", IEEE Tutorial: Distributed Processor Communication Architecture, October 1979, pp.18-23.

(5) P.H.Enslow,"Multiprocessor Organization - A Survey", Computing Surveys, Vol.9, No.1, March 1977, pp. 103-129.

(6) M.J.Flynn,"Some computer organizations and their effectiveness", IEEE Transactions on Computers, Vol.C-21, No.9, September 1972, pp.948-960.

(7) V.K.Ravindran and T.Thomas,"Characterization of Multiple Microprocessor Networks", COMPCON Spring, 1973, pp.133-137.

(8) D.P.Bhandarkar,"Analysis of Memory Interference in Multiprocessors", IEEE Transactions on Computers, Vol. C-24, No.9, September 1975, pp.897-908.

(9) D.P.Bhandarkar and Samuel H.Fuller,"Markov Chain Models for Analyzing Memory Interference in Multiprocessor Computer Systems", Proceedings of the First Annual Symposium on Computer Architecture, 1973, pp.1-6.

(10) J.R.Jackson,"Jobshop-like Queueing Systems," Management Science, October 1963, pp.131-142.

(11) W.J.Gordon and G.F.Newell,"Closed Queueing Systems with Exponential Servers," Oper.Res., 15 (1967), pp.254-265.

(12) S.D.Smith and H.J.Siegel,"An Emulator Network for SIMD Machine Interconnection Networks", Proceedings of the Sixth Annual Symposium on Computer Architecture, 1979, pp.232-241.

(13) R.H.Eckhouse,Jr.,J.A.Stankovic,and A.V.Dam,"Issues in Distributed Processing - An Overview of Two Workshops", IEEE Computer, January 1978, pp.22-26.

(14) Dijkstra,E.W.,"The Structure of the 'THE'-Multiprogramming System", CACM 11, 1968, pp.341-346.

(15) Dijkstra,E.W.,"Hierarchical Ordering of Sequential Processes", Acta Informatica 1, 1971, pp.115-138.

(16) Hansen,B.P.,"Structured Multiprogramming", CACM 15, 1972, pp.574-578.

(17) Hoare,C.A.R.,"Monitors: An Operating System Structuring Concept," CACM 17, 1974, pp.549-557.

(18) Campbell,R.H. and Habermann,A.N.,"The Specification of Process Synchronization by Path Expressions", Lecture Notes in Computing Science 16, 1974, pp.89-102.

(19) Habermann,A.N., Introduction to Operating System Design, 1976, SRA Inc., pp.348-352.

(20) Lagally,K.,"Synchronization in a Layered System", Operating Systems, Springer-Verlag, 1979, pp.253-281.

(21) Hansen,B.P.,"The Nucleus of a Multiprogramming System", CACM 13, 1970, pp.238-241.

(22) Lauer,H.C., and Needham,R.M.,"On the Duality of Operating System Structures", Operating Systems: Theory and Practice, North-Holland, 1979, pp.371-384.

(23) Opderbeck,H.,"Common Carrier Provided Network Interfaces", Operating Systems, Springer-Verlag, 1979, pp.482-516.

(24) Hansen,P.B., Operating System Principles, Prentice-Hall, 1973, pp.89-121.

(25) Kleinrock,L., Queueing Systems, Volume 1, John Wiley & Sons, 1975, p.104.

(26) IBM Synchronous Data Link Control, General Information, GA27-3093-1, May 1975.

(27) Walden,D.C.,"A System for Inter-process Communication in a resource-sharing computer network", CACM Vol.15 No.4, 1972, pp.221-230.

(28) Shoch,J.F., and Hupp,J.A.,"Performance of an Ethernet Local Network - A Preliminary Report", Compcon Spring, 1980, pp.318-322.

(29) Kleinrock,L.,"On Flow Control in Computer Networks", Proc. of International Conference on Communications, June 1978, pp.27.2.1-27.2.5.

(30) Chan,J.Y.K., and Georganas,N.D.,"Dimensioning of Message-switched computer-communication networks with end-to-end window flow-control", Sixth Data Communications Symposium, Nov.1979, pp.102-108.

(31) Schwartz,M., Computer-Communication Network Design and Analysis, Prentice-Hall, 1977, p.158.

(32) Metcalfe,R.M., and Boggs,D.R.,"Ethernet: Distributed Packet Switching for Local Computer Networks", CACM, July 1976, pp.395-404.

(33) Patel,J.H.,"Processor-Memory Interconnections for Multiprocessor", Sixth Annual Symposium on Computer Architecture, April 1979, pp.168-177.

(34) Tokoro,M., and Tamaru,K.,"Acknowledging Ethernet", Compcon Fall, 1977, pp.320-325.