BASIC RATIONALE FOR mP's

USER
- PERFORMANCE
- Δ PERFORMANCE AND RANGE
- AVAILABILITY VIA REDUNDANCY
  SERVICEABLE

MFG.
- COST (BETTER SPARES, MANUFACTURING
- LESS DESIGNS

BASIC IRRATIONALE FOR mP's.

- NO-PROGRAMS
- HIGH DEV. RISK.
- UNIPROCESSORS ARE MORE COST/EFF.
- BETTER PLAN
- IBM Rents.

$$t_c = 2t_r = 2t_w.$$

$$\frac{\#\text{ACCESSES}}{\text{SEC}} = \frac{\boxed{m}}{t_c}\left(1-\left(1-\frac{1}{m}\right)^{P}\right)$$

$$= \frac{m}{t_c}\left(1-1/e\right) \quad \text{for } m=\infty \text{ and } P=m.$$

$$= \frac{m}{t_c} \times .67.$$

PMS-Level Computer Architecture

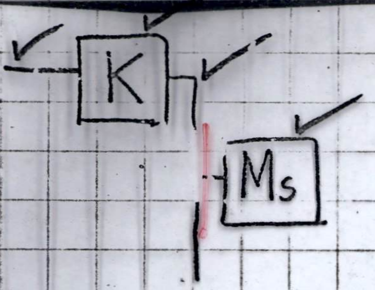
BASIC Problem: Specifying Load

User: Buying (Selecting);
Balancing (#M, P, T);
Designing Structures (Front ends, mP, closely Coupled, Net

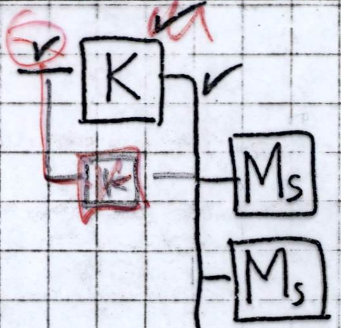
Manufactures:
New design structures;
Reliable, - PRESENT INTERFACE

Objective Fcn: Price, Availability, Performance, Size (Space & Power)

R-1



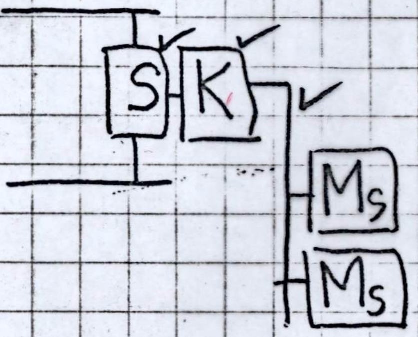
BASIC NEED

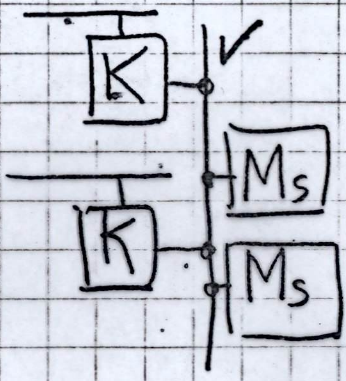


+ BACKUP DISK.

↙ PT. OF FAILURE

1C STRUCTURES

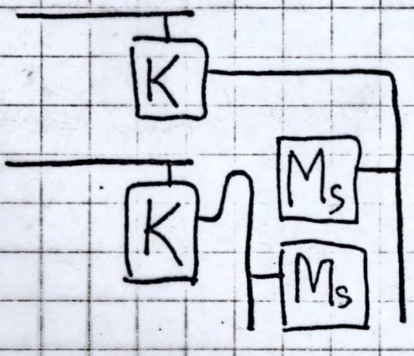


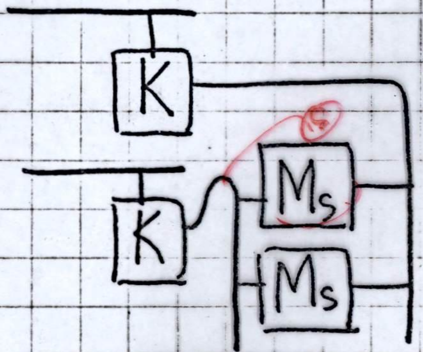


+ BACKUP C

2C (I.E. BACKUP)

+ BACKUP K.
(NO OFFLINE REPAIR)





(FAILURE OF IC -
DISABLES PATH TO
1 DISK)

+ 2 PORT Ms. -
WITH OFFLINE REPAIR

2 C - NO SINGLE PT. OF FAILURE

S-3090

MULTI-CPU Sig DECUS

MCPU SIG NEWSLETTER

MARCH 1977

Contributions and correspondence should be sent to:

Kevin C. McCue
MCPU Special Interest Group
% DECUS
146 Main Street, PD-3/E55
Maynard, Massachusettes

SPRING '77 SYMPOSIUM

Plans for the Spring Symposium in Boston are currently in progress, with sessions to include a SIG meeting, a Short Notes session, and a work-in-progress status update. If you would like to make a brief, informal presentation for the Short Notes session, or if you would like to help in some other way at the symposium, please contact the MSU SIG Chairman as soon as possible.

READERS' CONTRIBUTIONS

This section of the newsletter is devoted to articles submitted by the readers. Articles submitted by DEC personnel are submitted on an information exchange basis. They generally reflect one person's viewpoints, and do not necessarily imply any type of commitment by DEC.

CLOSELY COUPLED MULTIPROCESSOR SYSTEMS

Richard H. Eckhouse, Jr.
David L. Nelson

Advanced Development Group
Digital Equipment Corporation

March 1976

ABSTRACT

This paper presents the results of an effort to determine the performance, operational characteristics, hardware and software requirements, and the potential applications base for a symmetric system of closely coupled multiprocessors. Based on experience described herein, multiprocessing provides an effective way to increase the range of system performance with a single CPU product line, thereby serving a wider class of applications and market areas and providing explicit growth channels for applications whose computing requirements grow in time.

A prototype system has been built using PDP-11/40 processors, multiported memories, and UNIBUS windows, for the purpose of determining its performance and operational characteristics. The RSX-11M real time operating system has been modified to support multiprocessing on this configuration. Theoretical analysis has provided a mathematical expression for system throughput as a function of the number of processors, memory banks, and memory utilization factors. Performance measurements have been related to theoretical analysis so that analytic means can predict the performance of configurations beyond the scope of the prototype hardware.

For certain applications, the system cost-performance ratio is improved. The cost effectiveness of multiprocessing is contingent upon low processor/bus utilization of memory, or a high degree of parallelism in the memory system, such as interleaving or banking. Furthermore, realization of the potential afforded by multiprocessing hardware can only be attained in properly structured multiprogrammed operating systems.

I. INTRODUCTION

Multiprocessing provides an effective means of increasing the performance range of a single CPU product line. Increased performance through multiprocessing allows a mainframe manufacturer to reduce the engineering and support costs while serving a wide applications base. It does so because it is possible to offer greater system performance by combining multiple, lower performance processors. An equally important advantage of multiprocessor based systems is that they provide growth channels for applications whose processing requirements might increase in time.

Multiprocessing has been used in systems for the purpose of increasing both reliability and availability [1]. While these issues have not been directly addressed in this paper, the potential for greater reliability in similar systems occurs as a result of having multiple redundant processing units.

In certain applications, adding additional processors can increase overall system performance per dollar. These applications are generally restricted to those which are decomposable into many independant programs, and those in which greater computational speed for a single task is not strictly required. Response time is decreased in multiprocessing systems because processing resources are spread across several tasks, thereby reducing the processing rate of each (as compared to a larger single processor).

Definition Of A Multiprocessor System

By our definition a multiprocessing system is symmetric and operates with processors being treated as sharable resources without identities. In a symmetric system the executive floats from processor to processor. Consequently both user and system tasks run on any processor and which processor a task runs on is "transparent" to the task and the operating system. Conflicts in servicing requests are resolved by queuing up the requests. In order to operate in such an environment, the executive must be at least serially reuseable.

The advantages of a symmetric system over an asymmetric one are manyfold. First, a truly symmetric system can be expected to gracefully degrade as resources fail. Second, there is real redundancy because there are no "special" processors. Third, it is easier to make more efficient use of the resources because they may be pooled and used in an anonymous way. Finally, pooling of resources results in better avialability of them thereby resulting in a better distribution of the system load.

The disadvantages of a symmetric system are twofold. First, such a system may be more expensive because one cannot substitute lesser devices when they could replace their symmetric counterpart. Second, excessive system lockouts at the shared executive level may degrade system efficiency; consequently one has to pay particular attention to the placement and effect of software multiprocessor locks.

Hardware Requirements for Multiprocessing

There are at least three features needed to multiprocess in a symmetric fashion, using conventional DEC hardware. First there must be some form of hardware locking mechanism. Traditionally this mechanism is implemented by taking advantage of the read-modify-write cycle of some key instruction accessing main memory. Second, some form of interprocessor communication is necessary for the sharing of system tables and process information. This feature can be easily provided by either a common mapping of the information through the multiport memory, or by a memory management unit which can make the proper logical to physical memory translation. Third, an indexing or separation of "per-processor" information requires a processor identification. Some information must be associated with a particular processor, and the processor ID provides the index into this information.

Other desirable features include a) an interprocessor interrupt for an active form of interprocessor communication, b) a start/restart mechanism between processors, c) memory management for control of logical to physical mappings, d) interrupts passed to all processors within the system, and e) the ability to selectively arm and enable I/O devices. These features make it easier to schedule processes, exclude "failing" devices, and minimize system overhead. They are not required, however.

II. THE EXPERIMENTAL PROTOTYPE

In building the experimental prototype, we had to satisfy four goals. The first was our objective: the increased performance of a family of computers achieved by the parallel operation of identical processor components. The second was the application environment: supporting a conventional multiprogramming environment typified by our real-time system executives (RSX-11). The third was the method of implementation: utilizing standard processor and memory components of the PDP-11 family, allowing for incremental expansion without software modification. Finally, our fourth goal was methodology: developing a theoretical analysis which would allow us to compare the performance of the analytical machine with its actual implementation; equally important was the performance measurements of the prototype hardware and software.

Hardware Considerations

The initial multiprocessor configuration utilized in this prototype system is shown in Figure II.1. This quasi-symmetric system utilizes DEC's multiport memory, the MA-11 (manufactured by Computer Special Systems) , and a UNIBUS window, the DA-11F.

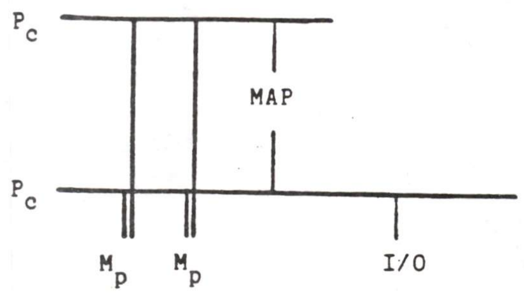


Figure II.1 -- Prototype Multiprocessor Configuration

The UNIBUS window has prevented us from building a dynamically reconfigurable, symmetric system, that is one where high availability is a design goal. In our system, only the second processor is allowed to fail, along with non-disk devices, if the system is to continue running.

III. THEORETICAL FRAMEWORK

In a formal sense, parallel processing hardware provides only the potential for increased system performance. Realization of this potential is attained only when the computing problem can be decomposed into smaller autonomous pieces that can be processed in parallel. Problem decomposition and parallel processing hardware are mutual requirements, therefore, in order to increase system performance.

In multiprogammed operating systems, problem decomposition occurs most naturally at the task level, primarily because programs are designed to process independently. It is a matter of convenience that the development of operating systems (designed primarily to protect, optimize, and share resources) has provided a decomposed computing environment amenable to multiple processors. We therefore expect to see multiprocessing to work well in applications which are multiprogrammed and where computing is primarily computation bound.

Problem decomposition occurs at other levels as well; for example, at the instruction level in a heavily pipelined single processor. Here, the degree of parallelism occurs internal to the CPU and works well in applications where computing is uniprogrammed as well as multiprogrammed. However, the approach is restricted to the extent that instructions are interdependent (branch on condition, for example) and to the extent that they address common registers and memory.

At levels between the instruction and task levels, problem decomposition becomes more complex. While multiprogramming requirements have provided task level decomposition, and instruction set architectures have provided instruction level decomposition, there is no direct analog at the intermediate operating system levels. Consequently, in order to realize multiprocessing potential for operating system functions, they must be designed to explicitly accomodate multiprocessors. (Note that in our RSX-11M prototype, operating system functions are not multiprocessed; rather, they are serially processed at the system executive level.) Fortunately, though, while the problems associated with parallel processing at operating system levels are more complex, the requirements for doing so are less important, since their functions are highly efficient in relation to the services (I/O for example) that they provide. Operating systems, therefore, are as likely to accomodate multiprocessing for generality as they are for performance.

Given this perspective, the analysis presented herein is relevant to multiprogrammed applications in which the computing requirement is primarily computation bound, and where parallel hardware acquires the form of separate autonomous, rather homogeneous, central processing units running out of shared memory.

Effects Of Interference On Performance

To a large extent, the effective performance of a system comprised of multiple processors is determined by the degree to which they interfere with each other. Interference occurs whenever the number of processor requests to a shared resource exceeds the number of allowed simultaneous accesses, thereby imposing a delay. Specifically, regarding multiprocessor systems, interference occurs whenever more than one processor accesses the same memory element. More generally, though, the analysis is extendable to software systems where software processes experience interference to the extent that they contend for shared programs and data.

This section discusses formal methods of determining the effects of interference on system performance for a rather general set of processors (or processes) operating in a rather general set of memories (or shared resources). Then, specific calculations are presented for a special case of symmetric processors accessing multi-ported, interleaved memory. The calculation is slightly generalized to include effects of cache memories so as to provide a framework for predicting multiprocessor performance of future machines.

General Description

Most of the research related to the performance effects of interference has involved digital simulations of computer models. Analytic approaches have not been as prevelant, primarily because of their complexity. We discuss here several important aspects of analytic approaches which range from being so complex so as to be insolvable in a practical sense, to a relatively simple closed form solution, each carrying different assumptions and levels of accuracy.

In general, a most important distinction between various theoretical approaches lies in the assumptions made regarding the time-ordered nature of processor to memory selection sequences. For example, in the most general case, each processor utilizes memory in a manner determined by a program which can be statistically modeled by assuming a probability distribution across the memory units. Here, though, if the distributions are uneven (processor utilizes one memory more than another), the solution becomes transcendental in that the effect of the interference perturbs the relative distribution itself, due to the fact that disproportionately more processor time is spent waiting for contended memories. By assuming an even distribution, the interference affects all processors uniformly, and the solution becomes somewhat simpler. An analogous situation is encountered regarding asymmetric processors.

Aside from the assumptions made about memory request distributions, there remain several approaches which can be characterized as to whether memory requests at one point in time are correlated to those at other points in time. Skinner and Asher [5] have taken this approach using a discrete Markov model, requiring all permutations of processor-memory requests to be explicitly analyzed - a formidable task for all but the simplest of configurations. The simplest analytic approach is to assume that each processor randomly reselects memory each cycle, regardless of whether it experienced contention during the previous cycle. Strecker and Bell [6] have taken this approach and derived a formal expression which is shown in good agreement with simulations. While our approach differs somewhat, the results presented herein are essentially the same. We derive here an equation for determining the throughput of a symmetric system of processors solely as a function of the number of memories, M, the number of processors, N, and the logical (excluding effects of contention) utilization of memory by the processors.

Case Of Symmetric Processors And Memories

In certain simple configurations, the calculation of interference effects can be obtained by appropriate analysis. For PDP-11/40 machines used in the multiprocessor prototype, and for future PDP-11 cached memory machines, we can assume the following model representation:

Pc - M.cache - S.Unibus - S.port - Mp

Let:

- r' = actual utilization (fraction of time) of memory by processor including interference
- r = product of logical utilization factors:
  - $r1$ = fraction of Pc time requesting memory

r2 = fraction of average memory time requesting backing memory,
r3 = fraction of backing store time that locks memory access.
M = number of parallel memory banks,
N = number of Pc's
T = throughput ratio of system (units of 1 Pc).

For each Pc cycle, the probability of a particular memory being selected is 1/M. Therefore, the probability that the memory is locked is r'/M, and the probability that the memory is unlocked is 1-r'/M.

For N Pcs, the probability that the memory is unlocked becomes

(1-r'/M)**N

so the probability it is locked is

1-(1-r'/M)**N

For N=M=1, the memory utilization (fraction of processor time that memory is locked) is r' = r. Therefore, the rate of memory cycles (throughput) normalized to that of a single processor single memory system is

(1/r)(1-(1-r'/M)**N)

and so the total system throughput ratio for all M's is

T=(M/r)(1-(1-r'/M)**N)   (Eq. III.1)

By approximating r' = r in the above equation, one obtains a relatively simple closed form expression which has been compared to more accurate forms (described below) to within ten percent. What follows is a derivation relating actual utilizations, r', to logical utilizations, r, as a function of T and N, thereby obtaining an accurate equation for T.

Letting t and tm be the average processor cycle time and memory locked time, respectively, and te be the average incremental time spent waiting for locked memory, then r and r' can be expressed as

r=tm/t;   r'=(tm+te)/(t+te)

which gives

te = (r' - r)t/(1 - r').

By our definitions, T/N is the ratio of the instruction rates of a processor with and without contention. The instruction rate without contention is clearly 1/t, and the instruction rate with contention is 1/(t+te), so that

$$T/N = t/(t + te).$$

Using the above expression for te, we get

$$T/N = (1-r')/(1-r)$$

giving

$$r' = 1 - (T/N)(1 - r).$$

Substituting this into equation III.1, and rearranging terms, we get an Nth order polynomial equation which can be solved for the system throughput ratio, T.

$$T-(M/r)[1-[1-(1/M)(1-(T/N)(1-r))]^{**}N] = 0 \quad \text{(Eq. III.2)}$$

<u>Determining r</u>

The effective utilization factor represents the fraction of processor cycle time that is spent in a contended (non-parallel) memory cycle. The numeric value for r can be obtained by multiplying the individual utilization factors of components that are accessed serially in time. Accordingly, we have defined r1, r2, and r3 as the fraction of time that (1) processor spends accessing the memory system, (2) fraction of time that memory system spend accessing backing store, (3) fraction of time that backing store is locked during a cycle, respectively. What follows is a brief analysis of each factor.

<u>Determing r1</u>

Measurements have been made to determine the fraction of time the processors accesses memory on PDP-11/40's by measuring the average bus cycle (1.8 microsec.), and the average amount of time that the processor is not accessing memory (.6 microsec.), giving

$$r1 = 1.8/(.6+1.8) = .75$$

This says that on average, the 11/40 spends about 75% of its time accessing memory - a fairly good balance.

<u>Determing r2</u>

The fraction of time the memory system spends accessing backing store is estimated by assuming a cache configuration THAT requires a full memory cycle on writes (10% of all requests) and a partial memory cycle on reads not found in the cache (13% of the remaining 90% of all requests), we obtain an average cycle time of

$$tave = .90(.87tcache + .13tmread) + .10tmwrite$$

For tcache = 300, tmread = 600 and tmwrite = 1100 nanoseconds,

$$\begin{aligned} tave &= .783x300 \text{ (cache)} + .117x600 \text{ (reads)} + .10x1100 \text{ (writes)} \\ &= 415 \end{aligned}$$

If we assume that the write portion of the read-to-core requests are completely overlapped with subsequent cache hits (so that the processor does not wait), then the fraction of memory time that requires the UNIBUS is

r2 = (.117X1100 + .10X1100) / 415 = .6

Determing r3

The fraction of time that backing store is locked has been obtained from measurements performed on a two processor 11/40 system having a single memory bank indicating an effective throughput ratio of 1.7 for compute bound jobs. Using the previously derived expression, we find that the value of r = .45 corresponds to the observed value of T = 1.7. Now, since the 11/40 is noncached, r2 = 1, and so we obtain the value of r3 (for the CSS MA-11) to be r3 = r/r1 = .45/.75 = .6.

Analysis Of Equation III.2

Equation III.2 has been numerically solved for the system throughput ratio, T, for several interesting configurations listed in Table III.1.

CONFIGURATION	r	T
two 11/40's:		
1 UNIBUS as memory switch	.75	1.3
1 MA11 memory ctl as switch	.75x.6	1.72
2 MA11's, interleaved	.75x.6	1.88
three 11/40's:		
2 MA11's, interleaved	.75x.6	2.63
4 MA11's, interleaved	.75x.6	2.84

TABLE III.1 -- T for Several Configurations

Comparisons of Equations III.1 and III.2 have shown that to first approximation, T is a function of the ratio M/r, rather independent of specific values of M and r. Consequently, for simplicity, we plot T as a function of M/r for several values of N, using a constant value of r=.45. Figure III.2 shows the relatively small dependence on r where T as a function of M/r has been plotted for several values of r using a constant value of N=4.

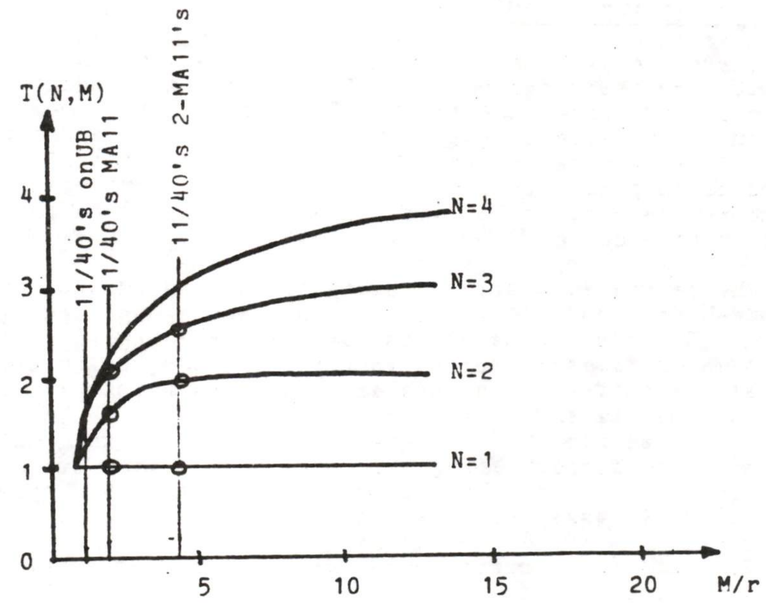


Figure III.1 -- T vs. M/r at r=.45 for several N

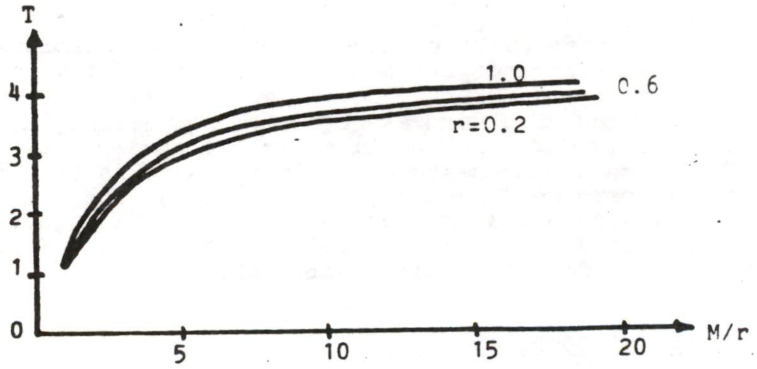


Figure III.2 -- T vs. M/r at N=4 for several r

IV. COST EFFECTIVENESS

There are two performance statistics which are important for our multiprocessor system. The first statistic indicates that when two processors are connected to the same multiport memory, and both processors are accessing that memory, then a timed program executes about 27% slower due to the second processors interference. This figure is surprisingly small, suggesting that processor utilization of the memory port is low and may not be greatly affected by the addition of a cache!

The second statistic consists of the throughput rates for both compute-bound and I/O-bound tasks running on our multiprocessor system. The job mix column is computed by taking the ratio of the uniprocessor times for completing the I/O task to the total time to complete each task in the system. The throughput rate is then the ratio of the elapsed time to complete all tasks on the uniprocessor to the elapsed time to complete all tasks on the multiprocessor system. The highest effective rate is 1.72 when two compute-bound tasks are running in a multiprocessor system. This rate drops to 1.01 when all tasks are I/O-bound.

To put this performance data into perspective, we must consider the costs of a multiprocessor system. A minimal uniprocessor system would consist of an 11/40 processor, 64K, memory management, a clock, console, boot loader and two disk drives. Adding a second processor without memory, but with a clock, bus window, multiport memory controller, and a cabinet/expander box would cost the end user 41% more. The result is an increase in performance ranging from 1% to 72% for a cost increase of 41%. To be fair, we should consider a more typical configuration rather than a minimal one. Using the more typical configuration, the addition of a second processor costs only 20% will yielding the same performance improvement.

Based on these figures, it is safe to say that for certain applications where tasks require additonal computational cycles the system cost-performance ratio will be significantly improved with a multiprocessor configuration. This cost effectiveness is contingent upon low processor/bus utilization of memory and decomposability of the application into separately computable tasks. The resulting performance improvement is then the result of processing resources being spread across the highest several priority levels so that the processing rate is reduced for the total application. Performance will not be affect for individual tasks but only for the entire application.

V. ISSUES YET TO BE ADDRESSED

If this prototype system is to be the basis for an actual product then it is necessary to decide where it is to fit into the real-time systems line, what it is that we wish to offer (e.g., greater performance and/or high availability), and which hardware and software changes to current systems we wish to make. Clearly the purpose of this research is to investigate which paths are possible to follow; it will not produce a marketable product. Nonetheless, the results obtained make it clear that multiprocessing is one way to gain greater system performance.

The issues remaining to be considered are a) improved system performance for I/O-bound tasks, b) distributed I/O to balance out system loading, c) a better understanding of real-time systems (gathering performance statistics and making models to predict system behavior), and d) actually measuring system performance for more than two processors and/or more than one multi-port memory. All of these issues are under consideration, although with somewhat lower priority than the original project, and will therefore be the subject of a later paper.

VI. <u>REFERENCES</u>

1) Enslow, Philip, ed., <u>Multiprocessors and Parallel Processing</u>, Wiley Interscience, 1974.

2) Wecker, Stuart, "A Building Block Approach To Multi-function Multiple Processor Operating Systems" AIAA Computer Network Systems Conference, April 16-18, 1973.

3) Wecker, Stuart, "Investigation of Multi-processor Mini-computer Systems", Research and Development Annual Report, Digital Equipment Corp., September 1973.

4) Wecker, Stuart, "A Design For A Multiple Processor Environment", IEEE COMCON, February 1973.

5) Skinner, C. and Asher, J., Effects of Storage Contention on Sysem Performance, IBM Sys. J., Vol. 8, no. 4, 1969.

6) Strecker, W. and Bell, C. G., An Analysis of the Instruction Rate in Multiprocessor Computers, internal DEC report.

DECUS
DIGITAL EQUIPMENT COMPUTER USERS SOCIETY
146 MAIN STREET, PK3/E55
MAYNARD, MASSACHUSETTS 01754
ADDRESS CORRECTION REQUESTED

BULK RATE
U.S. POSTAGE
PAID
DIGITAL EQUIPMENT
CORPORATION

1602-SPL-644
GORDON BELL
ML12/A51

Copy Dan & return

G. Bell
December 4, 1975

A Simple Example to Illustrate the Various Types of Systems

The following figures show solutions to the following general problem: A number of terminals, say 64, are located in various areas and need to access a central site where information is stored in a data base, contained on a single disk. Rather than totally pin down the problem more precisely, systems will be posited to show how various changes in the problem fit the solution. Although this is the proverbial solution looking for a problem, the purpose is to exhibit the conditions for the best solution, in terms of this basic problem. Also, it reflects the fact that a problem is likely to change over its lifetime by increased use; thus certain solutions tend to lock out solving future problems.

Depending on the relative costs of components (especially the communication lines), the hourly rate to not access the data, and the failure rates of various components, the optimum solution to the problem varies from a N, through a Cm, to an mP stucture.

Figures 1 through 17 show solutions to the terminal problem, and Fig. 17 gives measures of availability and performance for basic system price (excluding user and maintenance costs), for each of the configurations (and some variants).

Figure 1 shows the simple, basic solution with the 64 terminals connected via 4-16 K.comm (multiplexors). This is also most likely the cheapest solution to the problem, neglecting the communications line costs, which one might if the T's are hardwired and local to the configuration. It is also the most unreliable, and has the lowest performance.

If all 64 terminals must be up to be operational, then the meantime between failure for the whole system is only about 31 hours (2000/64). If we assume the terminal can be traveled to and repaired in only 5 hours, then the system availability is only 31/36 or 86%. Also assume the system is employed doing trivial operations, and the cost to not do the operation is only the cost of the people at the terminals (e.g., about $4/hour including overhead, etc.). The cost for each failure is thus 64 X $4 X 5 or $1280, which amounts to the cost of a single terminal, each time the system is down. Clearly operating in this mode is unacceptable and unrealistic, since there is no redundancy. Most likely, there would be extra terminals either on line, or nearby that could be placed on line rapidly as failures occurred. Clearly, the LA36's can and should be repaired on line.

So much for the terminals, let's look at the rest of the system. Assume the program is relatively small, and requires an 11/40 processor with 48K words of primary memory, i.e. 3-16K Mp's. The expected failure rate per thousand hours, for the system, except the disk is 3X.019+.06+.027 for Mp + Pc + Kdisk, giving a failure rate of .144, or 1 failure each 6940 hours. This is obtained by using the MTBF data from Table REL. Assuming it only takes 10 hours to find, call and fix the problem, the availability of this subsystem is 6940/6950 or 99.8%. Now adding in the disk brings the failure rate up to .544 failures per 1K hours or an MTBF of 1838 hours with an availability of 1838/1848 or 99.4%.

The cost of the disk failures are about 10 hours X64 X $4 or $2560, and the 2500 hour failure rate is about 1 year, hence the cost due to lost time due to the disk is 2.56K/year. This assumes that a disk failure is not of a simple catastrophic nature when it fails, and then is repaired with no loss of work. Several days of work could be lost due to faults, and the cost per year could exceed the disk cost. It is desirable to have a second disk.

Disk Redundancy

Figure 2 shows the 2 disk configuration. Assuming we absolutely want information recorded permanently, adding a second disk requires more Pc time, and the system performance is degraded, because everything must be written twice. The system costs more, but the reliability now approaches that of the Pc and Mp, i.e. 6940 hours, or about 3.5 years. Here, the disk reliability has been increased such that it can be neglected. The probabilities are:

$$p(\text{disk down}) = \frac{10}{2500} = .004$$

$$p(\text{at least 1 up}) = p(1\text{ up}) \times p(2\text{ down}) + p(1\text{ down}) \times + (2\text{ up}) + p(1\text{ up}) \times p(2\text{ up})$$

$$= 2 \times .996 \times .004 + .996 \times .996$$
$$= .999984$$
$$= 1 - p(\text{both down})$$
$$= 1 - .004 \times .004$$
$$= 1 - .000016$$

$$\text{MTBF} = \frac{\text{repair time} \times P(\text{up})}{p(\text{down})} = \frac{10 \times .999984}{.000016}$$

$$= \frac{10 \times 999984}{16}$$

$$= 624{,}990 \text{ hours}$$

Actually, there are other variants of the structure that might provide a similar benefit, but at a reduced cost and/or performance penalty. A tape would permit the disk entries to be back-up and recreated. Alternatively, a second disk need not be an exact copy of the first, but rather, it would be updated periodically to reflect the latest version of information. In this way, disk data need not be continuously written and the performance is improved.

By getting rid of the disk and terminal reliability problems, the rest of the structure must be considered.

Figure 3 is a 2 Pc, multiprocessor structure and this redundancy is the simplest way to back up failing parts: a communications controller, a disk controller, another primary memory module and finally an additional processor. The UNIBUS generally permits structures of this form to be built, although a second Pc cannot be added in this fashion, although it could be (a detail which will be ignored for now).

With this multi-Pc system, the component failing the most often is the Pc, hence, adding a second Pc would significantly increase the reliability. Here, however, one has an opportunity to use a different strategy by using smaller Pc's. If we can use an 11/04 or 11/05 Pc, at lower performance than the 11/40, the reliability is greatly increased. Otherwise, there is a loss, since the reliability for several small Pc's is worse than a single larger Pc. The actual reliability of the Pc only is quite high as seen from Table REL; the cabinet, bus (connectors), power supply determine the reliability.

Cfe, Front End Computers

A network can be formed by adding another computer, Cfe (see Figures 4 and 5), to handle the front end, communications processing load. We can see intuitively that the system is significantly less reliable. Another C has been added and the only way that we might expect reliability to be any better is that by having less components and software in Cmain, its reliability is somehow much better to cover the loss of reliability when Cfe is added. This is usually the argument for functional specialization, although since we are not considering the reliability of software, it's difficult to make the argument.

To a first approximation, the issue of a Cfe is almost orthogonal to the questions at hand. Why then add the Cfe?

1. There are a significant number of terminals residing at remote sites such that concentrating messages remotely saves line charges.

2. The Cfe being added makes a negligible decrease in availability.

3. Cmain gets overloaded, hence the cost to add capacity is very high, compared to a single, functional component, Cfe.

4. There are too many lines coming into a single Cmain, such that the reliability is impaired.

5. Cfe can act as a switch, S, to one of several Cmain's.

6. The problem can be broken up and solved on a functional basis, thus increasing the reliability, availability, flexibility of both parts.

Terminal Redundancy

Figures 6-11 examine the problem of increasing the component redundancy associated with terminals such that this information is more likely to reach the main computers. A key aspect of the front end redundancy problem is associated with the location of the terminals. We assume that the terminals are located at a single site (or arrive through a single telephone exchange) because they either provide redundancy of multiple K.comm's or switch a single K.comm to 1 of 2 local computers. In Figures 6-7 a duplicate (redundant) set of K.comm's provide an alternative path to either of two C's. Figures 8-10 show a single K.comm which is switched to 1 of several C's.

In Figure 6 each Cfe (or pair of Pc's) has its own independent set of K.comm's such that terminal (via its communications line modem) can send information to either one of the two K.comm's. Such a structure can be build by modifying communcations modems to feed two independent controllers either via a bussing arrangement or by switching at the modems. This structure provides for the highest reliability since either K.comm can operate the communications link and there is no extraneous equipment between the line and K.comm's.

Although logically identical, a switching arrangement of this type permitting a communications line to be sent to either of the two independent C's can be provided in the communications subsystem (Fig. 7). Here, we assume that either computer only uses an active line, and the lines can be distributed somehow between the two computers. In some systems this switch is automatic, but it could be manual as a single plug-type switchboard.

Note that a switchboard is most likely used without complete duplicated Cfe and is perhaps the most realistic (useful) system in view of the high reliability of K.comm and the Cfe (particularly the smaller size). Alternatively we need not replicate the controller section of a Cfe, but yet provide backup to the Cfe, by a wholly replicated Cfe, as shown in Figure 8. Note, we got to a duplexed Cfe because it was necessary to backup the entire front end computer. While this looks extravagent when there is only a single Cfe, but with n Cfe's, adding a single Cfe is only a 1/(n+1) increase in size for redundancy. Figure 8 with a spare Cfe, is the most likely structure when there is a substantial number of Cfe's. Note, having to route n Cfe's to a set of Cmain's, creates a significant problem with Cfe X Cmain interconnections. Ultimately this leads to a separate store and forward network structure based on packet switching to get the Cfe's connected together and to the Cmain's...but that will be covered later. The other key advantage of having a fixed K.comm structure is that when Cfe's are used, they can be located at the site with the terminals. In general, the structure of Figure 8 is more cost-effective (even with no redundant input or switchboard) than Figures 6-7; and since the K.comm's are unlikely to fail, the following structures with single K.comm's will probably tend to be used.

Figures 9-11 are also single K.comm structure but with various kinds of switching to the alternative computers. In this way only a single controller is required and can be switched to either 1 of 2 C's. The problem with such a structure, however, is that switching can be quite large, costly and unreliable; hence the actual reliability for a single shared K.comm, can be lower. Figures 9 and 10 show a K.comm connected to 2 and 3 C's respectively.

Figure 11 has come to be the most useful method of switching and is called the Unibus Switch, i.e., S(Unibus). With it, a Unibus, with 1 or more controllers are connected (if only 1 then we have Figure 9) in a group to 2 local C's. In this way, the C's are backed up but there is no redundancy in the K.comm's. Also we have added the unreliability of the S(Unibus), and there is no way to use both Cfe's under heavy load conditions. This, of course, can be partially helped by using two Unibus Switches and attaching 1/2 K.comm's to each switch. The Unibus Switch would only be used when the reliability of the part being attached is significantly higher than the rest of the system being attached to.

The Network-Cmain Structure

The structures of figures 12-14 are designed to examine the alternative structures: N - Cm - mP, respectively. Figure 12 requires duplexed links to carry traffic, to communicate exhaustively with both Cmains, and for reliability. Here, we have assumed the terminals are distributed in two remote sites with the Cmain at a third, neutral or central site (actually the 3 sites could be merged to 1). As the terminals become concentrated in a single site, and the computing (performance) remains high, a Cm structure can be used - and the Cfe's can be directly connected to Cmain's as in Figure 13 using high speed links. When additional Cfe's are added, two links from each Cfe would be required to each

Cmain. Hence, we begin to see a more centralized structure (e.g., mPc) is easier to exhaustively connect. This scheme works fine until a large number of Cmains are used - as discussed in the front end section above.

The issue to use either of these structures is fairly straightforwared, for fundamentally they are, or can be transformed to be the same just by moving them closer and affecting the data-rates and the locality of the interconnection. The previous discussion discussed the Cfe at a remote versus a central site, together with the attendant performance and reliability. The tradeoffs are just:

	N	Cm	Implication
Degree of Coupling	low	high	(permits Cfe-Cmain tradeoff)
Remoteness of T's	yes	no	(saves T link costs)

Figure 13 permits more lattitude in moving functions between Cfe and Cmain, because there is very high bandwidth between the four machines. By having a remote Cfe, and low data rate links to Cmain, the problem must be clearly segmented so that a high degree of interaction is not required. Similarly, as terminals are needed at remote sites, a remote Cfe becomes an economic necessity. DECNET software protocols permit N-type and Cm-type computer interconnection on a transparent basis. The interconnection of a common Ms.disk, however, is not included in the basic DECNET software. Therefore Cm's and Networks, neglecting the shared Ms.disk for Cm's, are handled identically in DECNET software. The Cm's enable applications not handled by DECNET.

Cm_and_mP_Structures

Just as we showed that Cm and Network structures were highly related and differed only in the degree of coupling, we can show that multiprocessor structures are quite closely related to computer module structures and represent further increase in the degree of coupling. By redrawing Figure 13, it can be easily transformed into a multiprocessor. Figure 14 is Figure 13 redrawn, where each computer of the Cfe and Cmain is expanded into its constituent Pc and Mp parts. Notice the duplexed links from the 2 Cfe's to the two Cmain's, and the link between the two Cmain's. Figure 14 also shows closeness of coupling between the 4 computers. Now, by a very simple transofrmation on Figure 14, a 4 Pc, multiprocessor can be formed by increasing the degree of coupling among the Mp and Pc using a multiport memory (see Figure 15).

In the structure of Fig. 15, the links among the C's of Figure 13 are removed, and replaced by the multiport memory, thus achieving the highest degree of coupling...in effect, infinite bandwidth since a block of memory can be transferred among computers in Ø time. The gain is to have substantially less memory, since the operating system need not be replicated in the 4 modules...in essence, the hardware switch (overhead) is introduced to share memory. As for the Pc interconnection to the various K.comm's and K.disk's any number of connection schemes can be used, and it is relatively irrelevant which one is used, assuming all Pc's are about the same performance. The only requirement is to not connect all K's to a single Pc. A reasonable interconnection strategy would be to connect 1 K-type to each Pc. By doing this, note, we have nearly come full circle, and formed two Cfe's and two Cmain's, except that there is a higher degree of coupling, and any task in the Mp can be run on any Pc. The main advantage of the multiprocessor structure is that if only a single Pc is required to do the job, then, 2 Pc's can be provided (for redundancy), but giving twice as much power as needed such that the computing requirements can expand. Also, as computing requirements expand, up to 3 more Pc's can be added and still have a spare capacity.

In summary, the tradeoffs of Cm (e.g. Fig. 14) versus Mp (e.g. Fig. 15) structures are:

	Cm	Mp	Implication
Degree of memory coupling	high	infinite	shared
Performance	f(Mp interference, and intercommunication among processes)		
Reliability parts)	high	higher	(note less
Memory size	--	lower	shared Mp
Cost	f(memory size and memory switch cost)		

As it is required that Cfe remote Cfe's are added, due to communications link costs, a multi-Pc structure can still be used, as in Figure 16. Here, note that only 2 Pc's are shown, but again, up to 4Pc's could be added as means of growth. Figure 16, is essentially identacal to the original network structure of Figure 12, except that the 2 Cmain's are simply replaced by a single 2 Pc multiprocessor, where less shared memory and switching of MPc is traded off against explicit interconnection of Cm. Note the multiprocessor achieves significantly greater reliability by removing the Mp from the Pc, with less Mp (each with an independent power supply and cabilnet system). This makes Pc smaller thereby increasing its reliability.

Figure 17 regresses back to the structure formed by combining the 2 Pc(1 bus) and the Cfe's of Figures 3 and 5. It points out that while we can obtain some of the reliability and performance capabilities of the multiprocessor, it's limited by the single, shared bus. Note, as in Figure 3, there are redundant components for all except the single bus. in contrast, the multiprocessors of Figures 15 and 16 have no shared components except parts of the Cfe (which can be backed-up as discussed in the previous section).

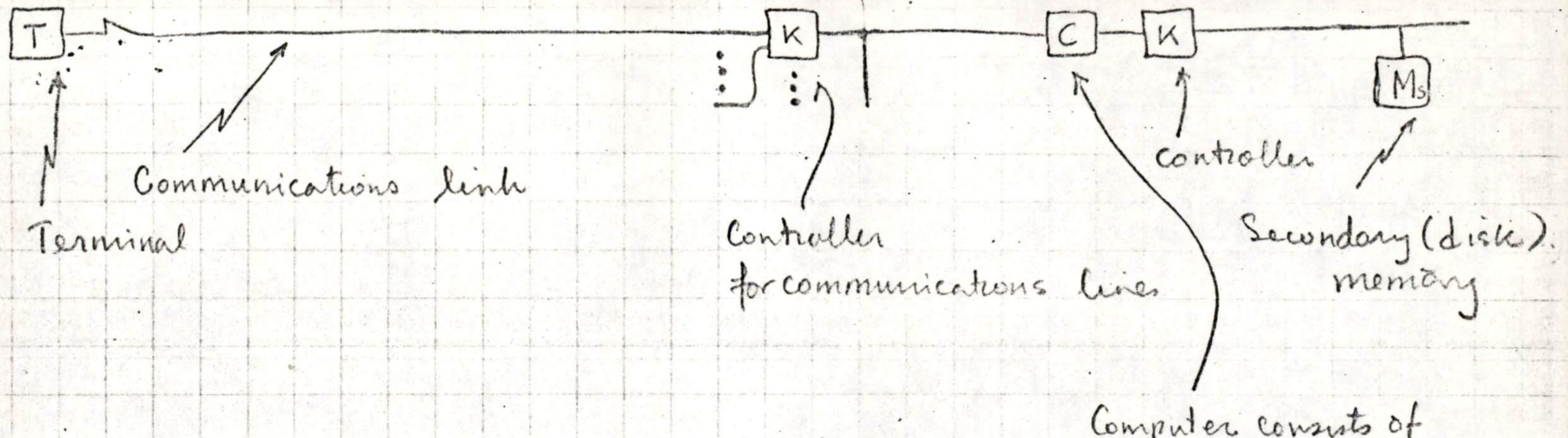

T
K
C
K
Ms
Communications link
Terminal
Controller for communications lines
controller
Secondary (disk) memory
Computer consists of Processor (Pc) + Primary Memory

Fig. 1 Basic C, 1 site

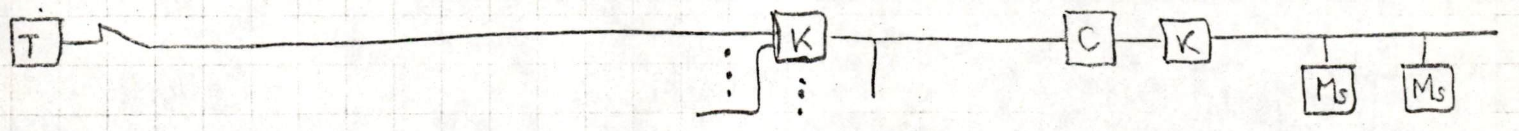


Fig. 2 Basic C with redundant disk, 1 site

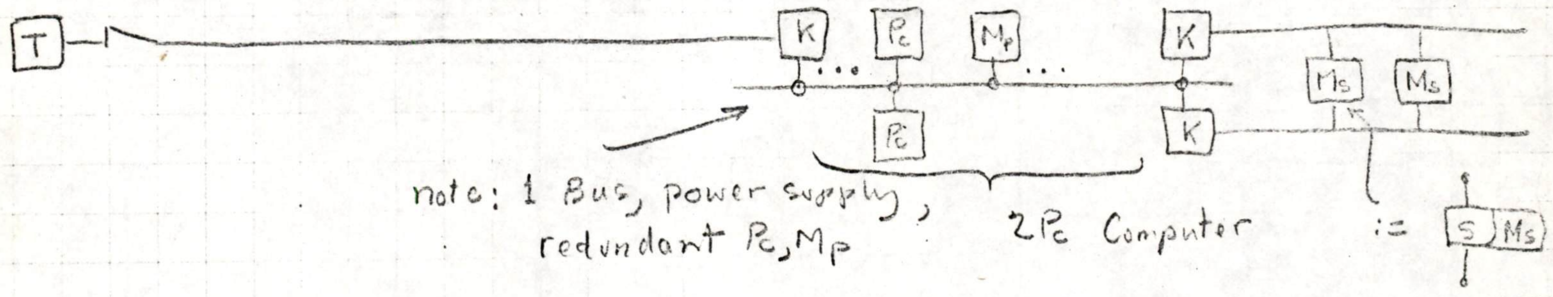


Fig. 3 2 Pc, 1 bus C, 1 site

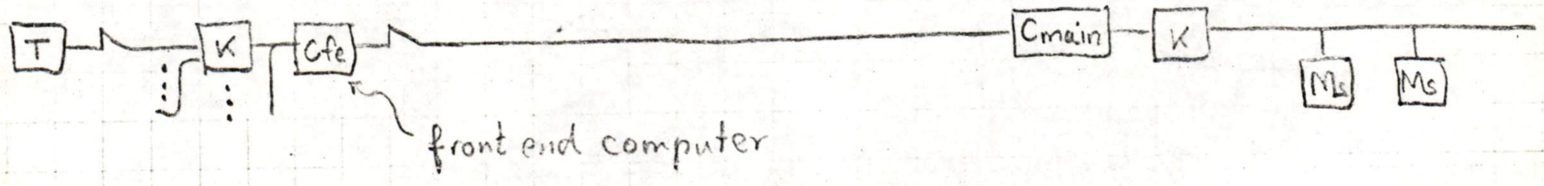


Fig. 4. N := (Cfe + Cmain), 1 site

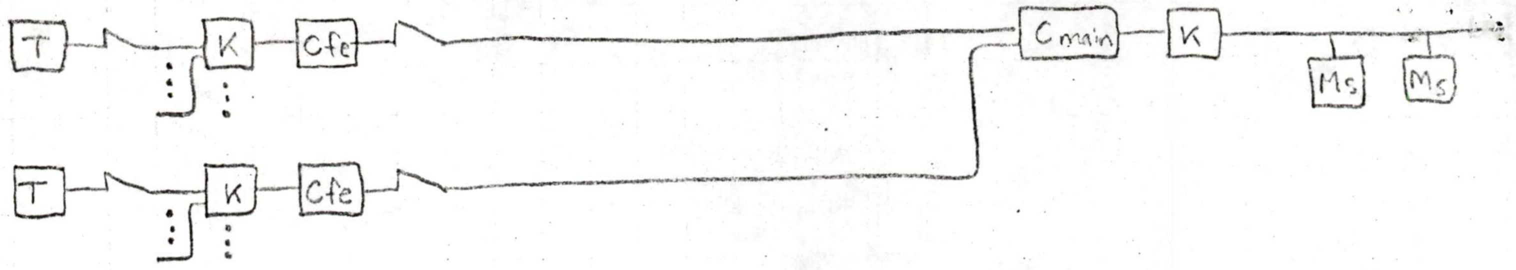


Fig. 5   N := (2 Cfe, at 2 local sites + Cmain), 1~3 sites

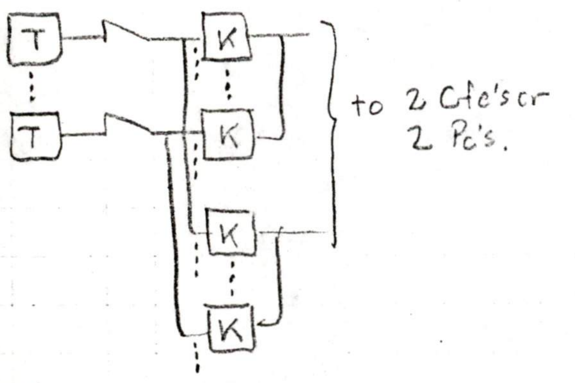


Fig. 6   Bussed T's connected to redundant K.comm's, 1 site

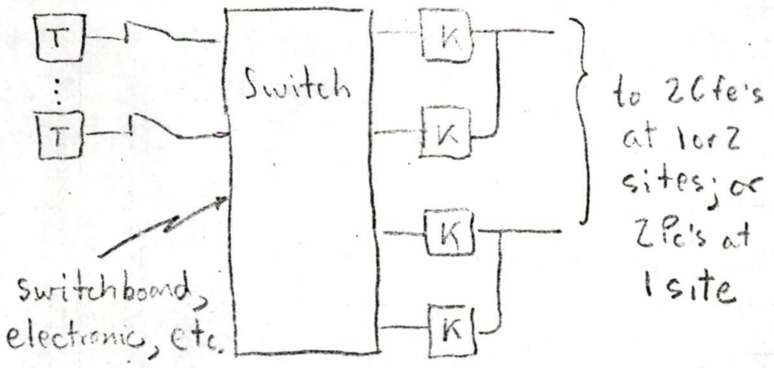

T
T
Switch
switchboard, electronic, etc.
K
K
K
K
to 2 Cfe's at 1 or 2 sites; or 2 Pc's at 1 site

Fig. 7   Duplex switch to redundant K.comm's, 1 or 2 sites

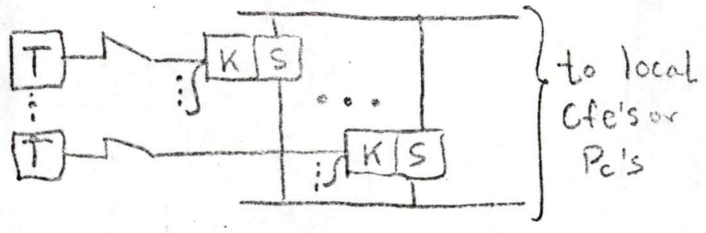


Fig. 9   Duplex switch at each K.comm to redundant Cfe's or Pc's, 1 site

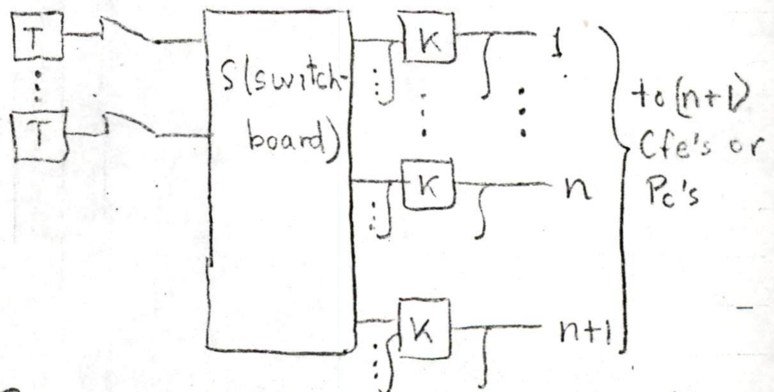


Fig. 8   Switchboard to a redundant Cfe or Pc at up to n+1 sites

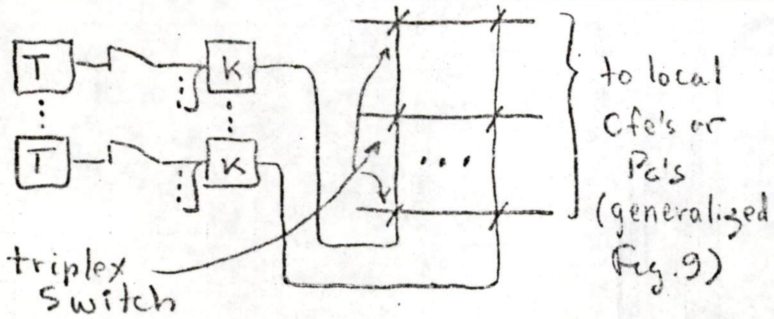


Fig 10.   Triplex Switch at each K.comm to 3 (Cfe's or Pc's) at 1 site

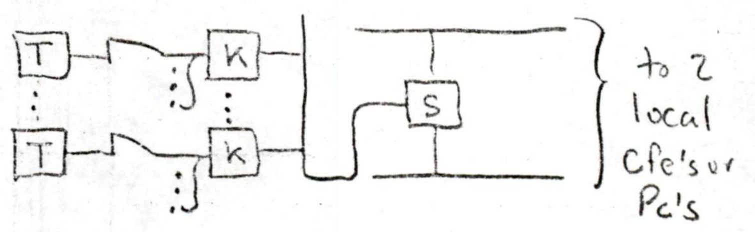


Fig 11.   Unibus switch for single K.comm's to redundant Cfe's or Pc's at 1 site.

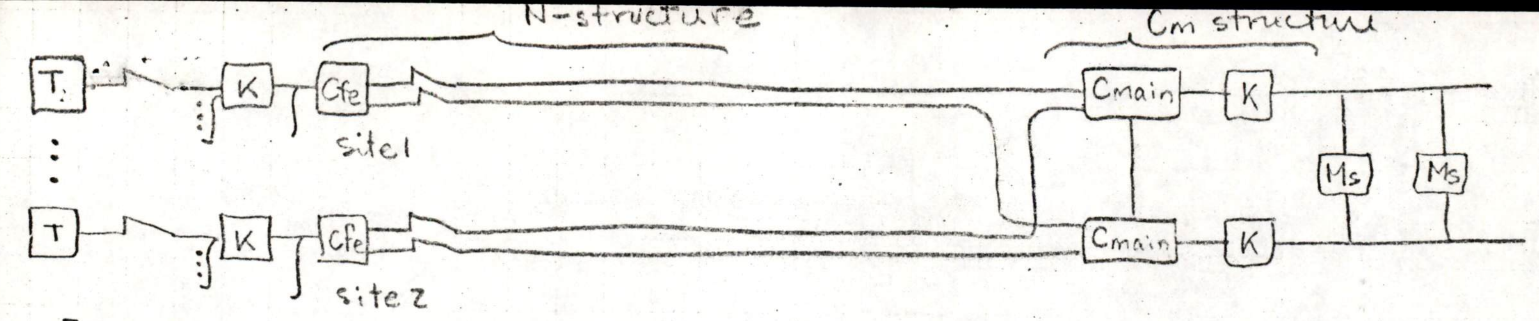


Fig 12 N-Cm structure, 1~3 sites

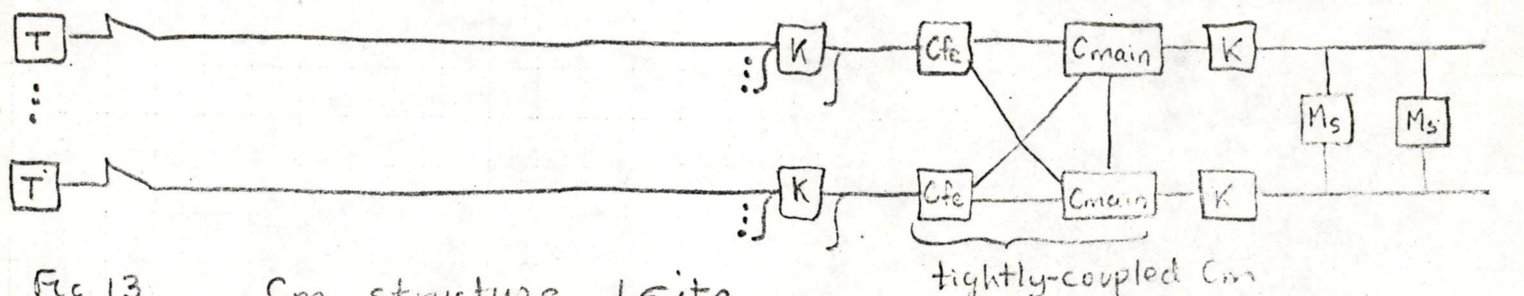
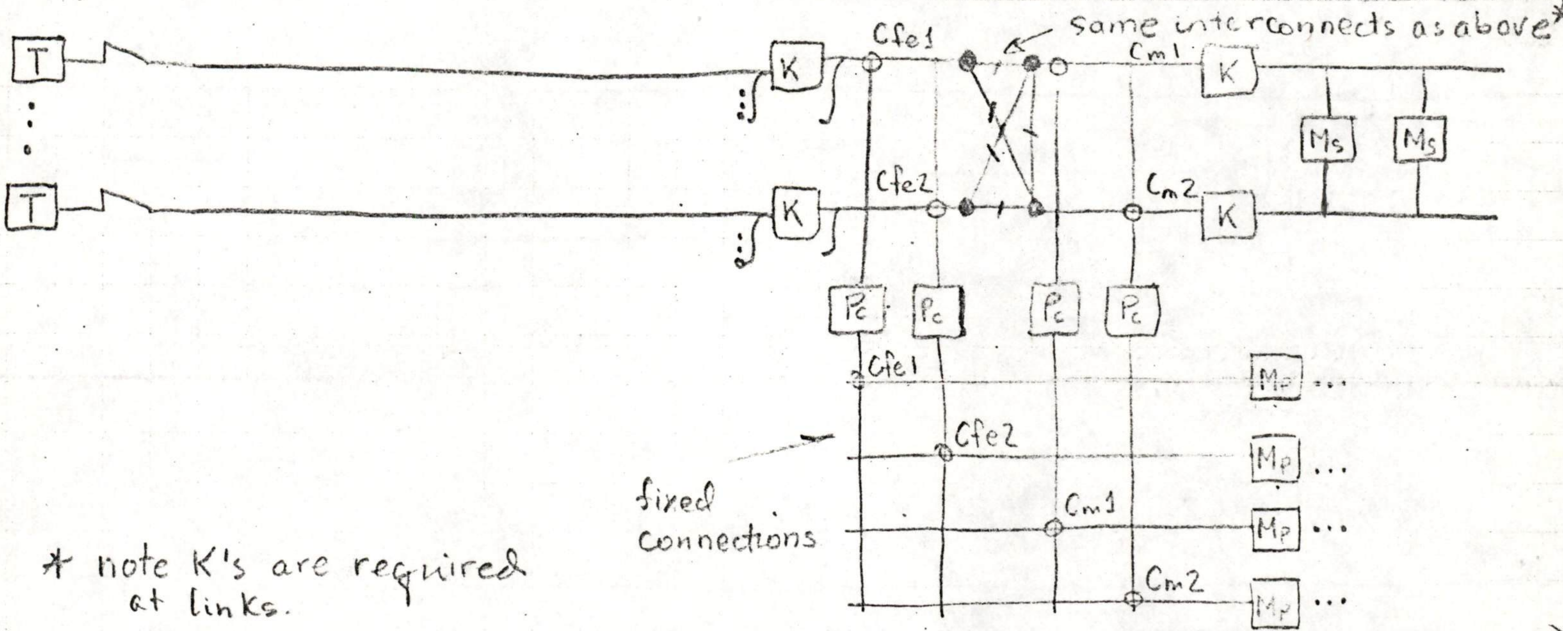


Fig 13 Cm structure, 1 site



* note K's are required at links.

Fig 14 Cm structure, 1 site (Fig 13 redrawn as a potential multiprocessor)

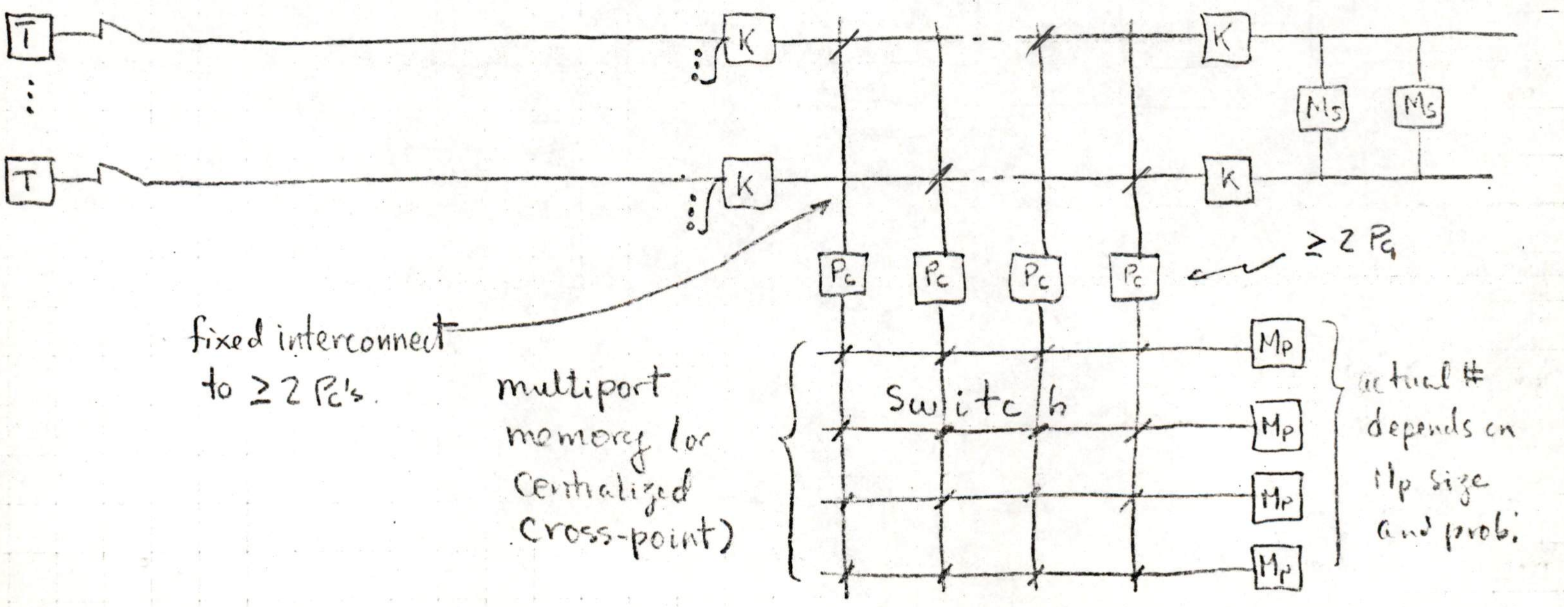

T
K
Ms
Pc
≥ 2 Pc
fixed interconnect to ≥ 2 Pc's
multiport memory (or centralized cross-point)
Switch
Mp
actual # depends on Mp size and prob.

Fig 15 4 Pc (multiprocessor) structure, 1 site

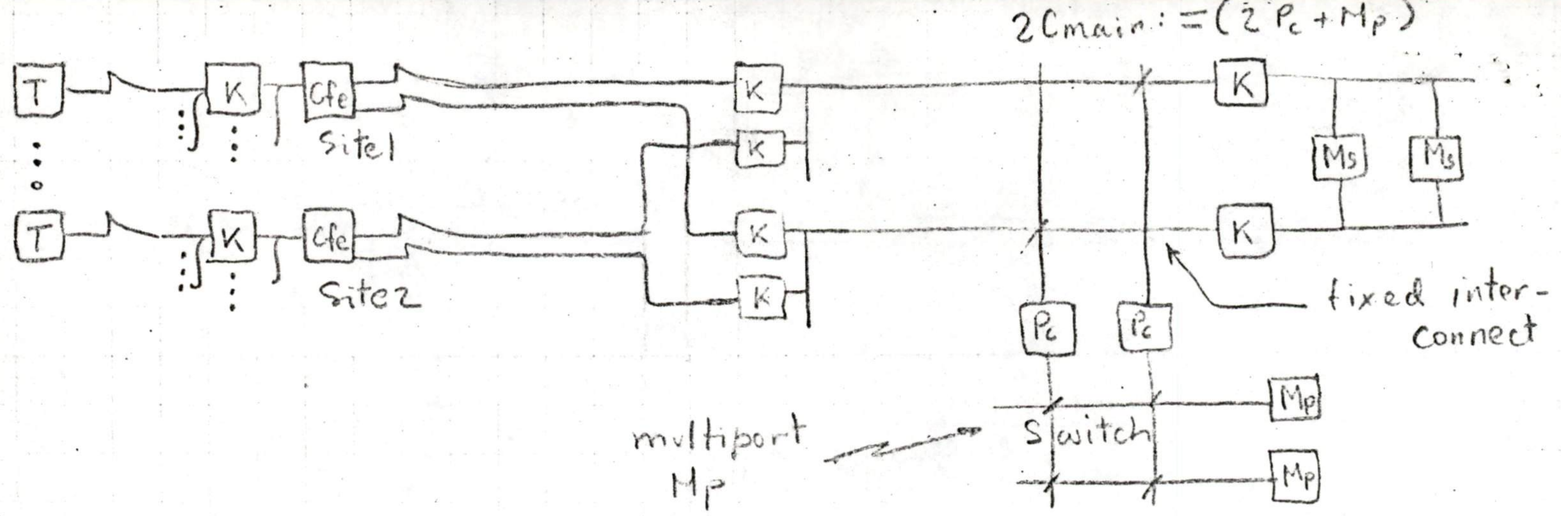


Fig. 16 N - 2Pc structure, 1~3 sites

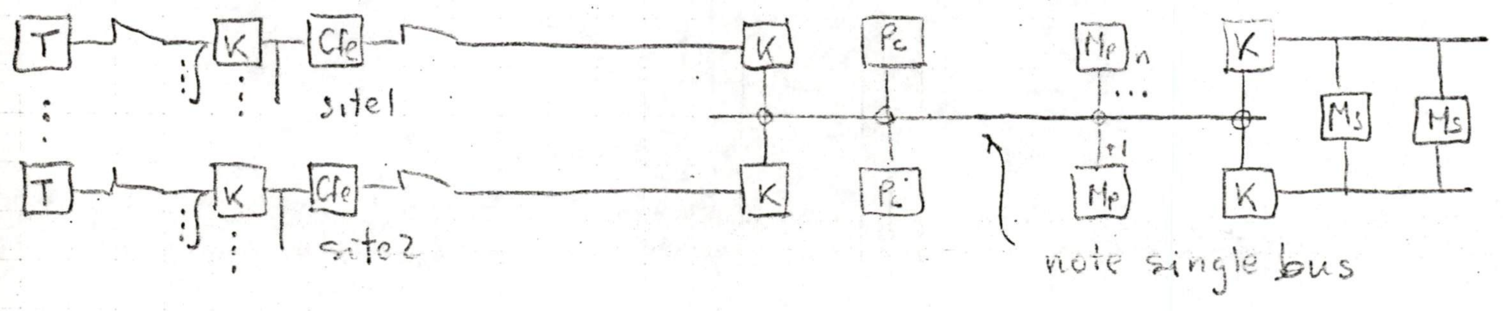


Fig 17 N - 2Pc (1 Bus) structure, 1~3 sites

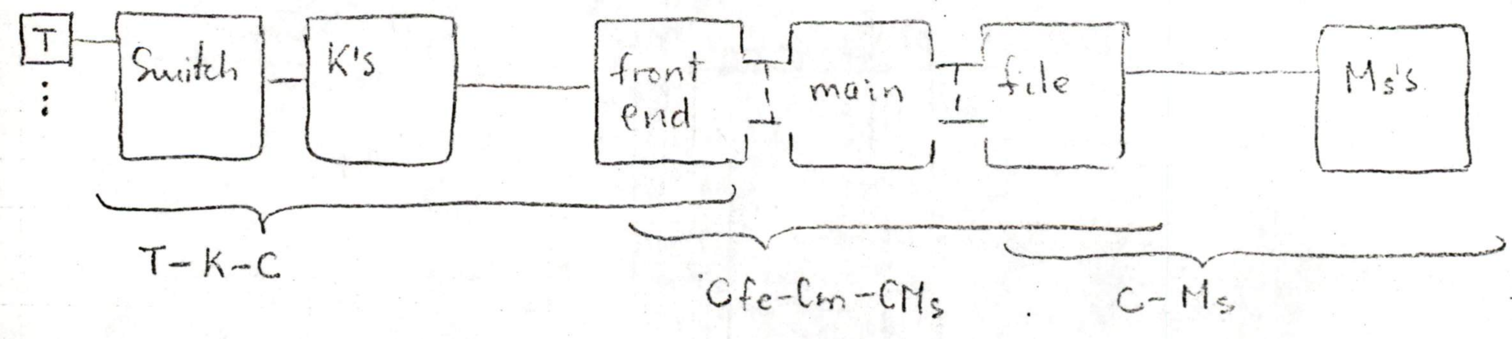


Fig 18. Problem interconnection / structure.

Assumptions to ~~[illegible]~~ compare various structures

In order to make ~~a~~ quantitative comparisons among the various structures, ~~we must make~~ some assumptions must be made about the Mp module size and the ~~together with~~ ~~the~~ requirements ~~the~~ (operating system and problem).

These assumptions are:

Operating System and Problem size (in Kwords)

main + fe (in a single system)	20
main + buffers for a remote fe	20
fe alone	8
fe buffers	4K / 16 lines
~~32K~~ main problem	30
Therefore:	
fe with 32 lines	16
" " 64 lines	24
Single system with all requirements	66

Mp module size & current products (in Kwords)

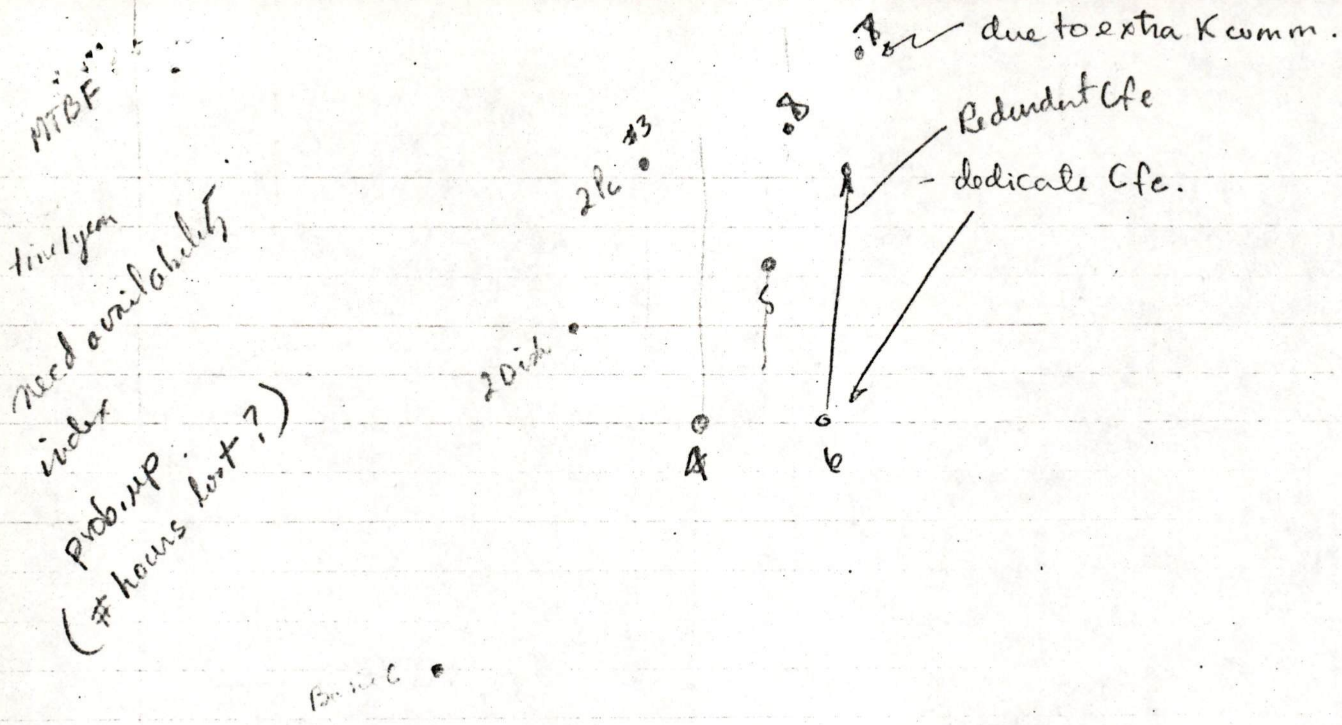
-Single-port	16
-Multi-port	16, 32, 48, 64

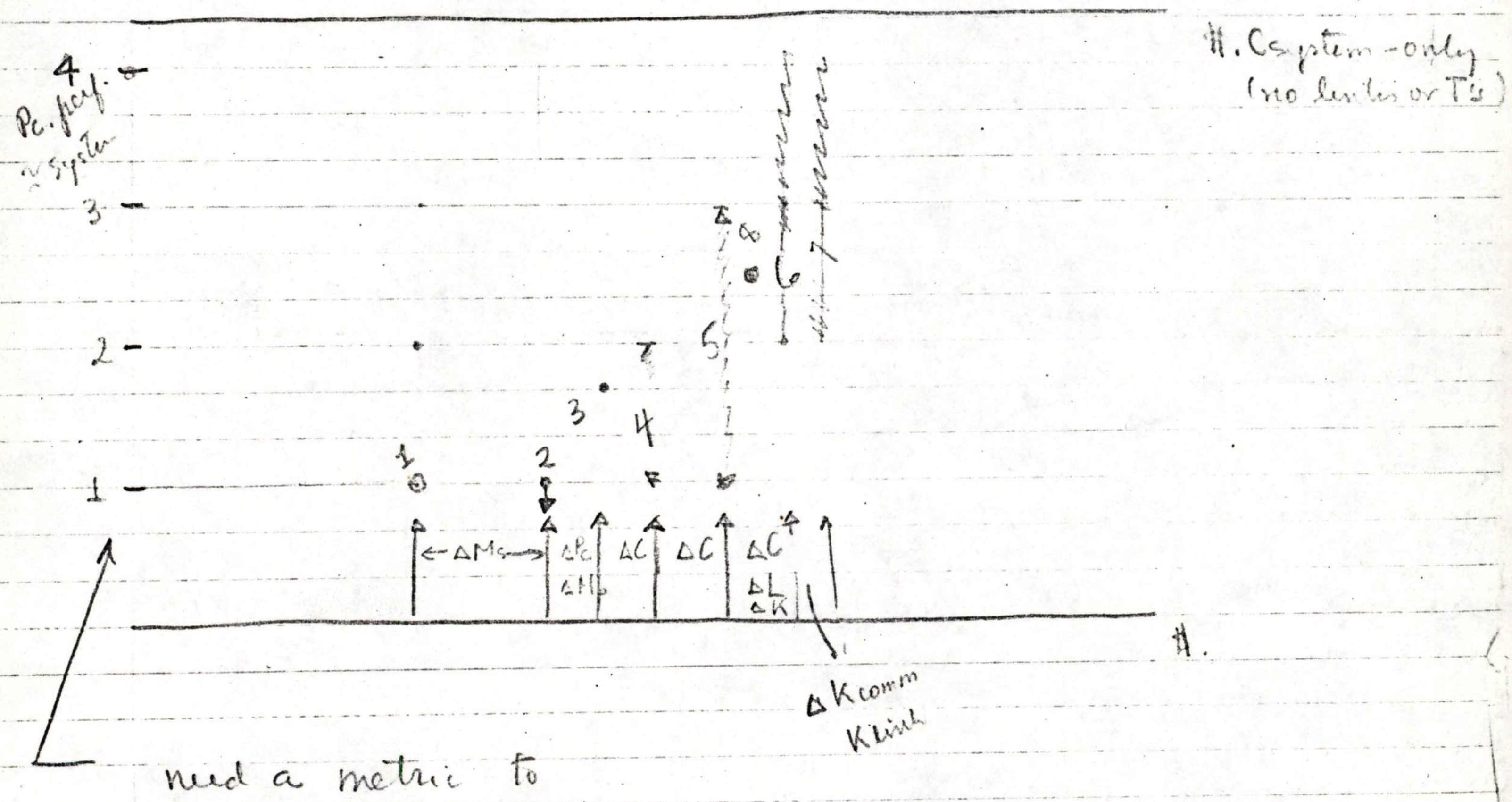
	T	Comm.			Ms		C			Sites	Type
	T	K.cn	L(High)	S.cn	Ms	S(Ms)	Mp*	Pc	S(Mp)		
1	64	4	0	0	1	0	66 / 4*	1	0	1	Uniprocessor
2	↓	4	0	↓	2	0	66 / 4	1	↓	1	"
3	↓	4	0	↓	↓	1	66 / 4	≤2	↓	1	2P multiprocesso
4	↓	4	1	↓	↓	0	72 / 5	≤2	↓	≤2	2C network
5	↓	≥4**	2	↓	↓	0	82 / 5	≤3	↓	≤3	3C network
5+6	↓	8	2	0						L	
5+7	↓	8	0	S(L)						1-2	
5+11	↓	≥4	0	S(UB)						1	
12	↓	≥4	1+4 ***	0	2	1	132 / 8	≤4	0	≤3	4C, Network-Multi-C
13	↓	4	0+5	↓	↓	↓	132 / 8	≤4	0	1	4C, multi C
15	↓	4	0	↓	↓	↓	66+ / 6	3.3	≥3	1	4P multi proc.
16	↓	≥4	4+0	↓	↓	↓	82 / 7	≤2+1.7	≥3	≤3	3C, Net - Multipr
17	↓	≥4	2+0	↓	↓	↓	82 / 6	"	0	≤3	", " "

* assumes 32K / module for multiport; 16K for single port (gives Kwords and # 16 K modules)

** depends on dist of T's

*** L(comm) + L(inter-C).





need a metric to reflect additional Kdisk on performance

Fig Availability and Pc. performance versus System price (excluding maintenance)

	T	n.cn	L(Ugh)	S.cn	M_s	$S(M_s)$	M_p*	P_c	$S(M_p)$		Sites	Type.
1	64	4	0	0	1	0	66 4*	1	0		1	Uniprocessor
2	↓	4	0	↓	2	0	66 4	1	↓		1	"
3	↓	4	0	↓	↓	1	66 4	≤2	↓		1	2P multiprocess
4	↓	4	1	↓	↓	0	72 5	≤2	↓		≤ 2	2C network,
5	↓	≥4**	2	↓	↓	0	82 5	≤3	↓		≤ 3	3C network
5+6	↓	8	2	0							1	
5+7	↓	8	0	S(L)							1-2	
6+11	↓	≥4	0	S(UB)							1	
12	↓	≥4	1+4 ***	0	2	1	132 8	≤ 4	0		≤ 3	4C, Network - Multi-C
13	↓	4	0+5	↓	↓	↓	132 8	≤ 4	0		1	4C, multi C
15	↓	4	0	↓	↓	↓	66* 6	3.3	≥3		1	4P multi proc.
16	↓	≥4	4+0	↓	↓	↓	82 7	≤2+1.7	≥ 3		≤3	3C, Net - Multipr
17	↓	≥ 4	2+0	↓	↓	↓	82 6	"	0		≤3	", " "

assumes 32K / module for multiport; 16K for single port (gives Kwords and # 16 K modules)

depends on dist. of T's

* L(comm) + L(inter-C).

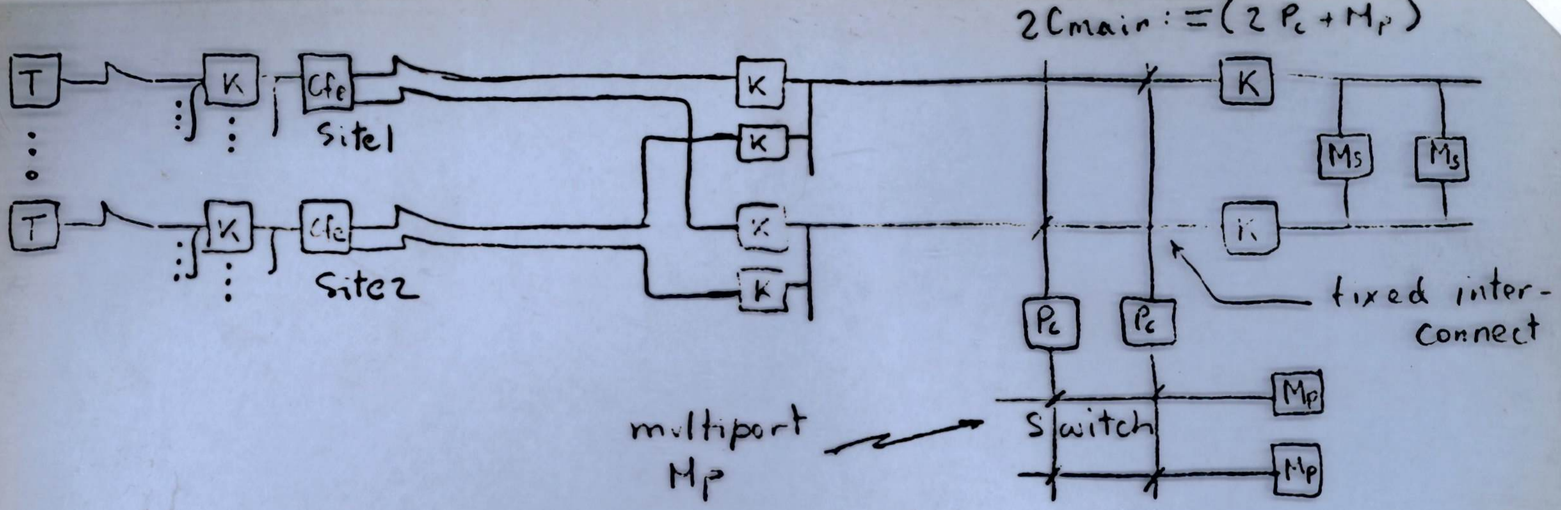


Fig. 16 N - 2P_c structure, 1~3 sites

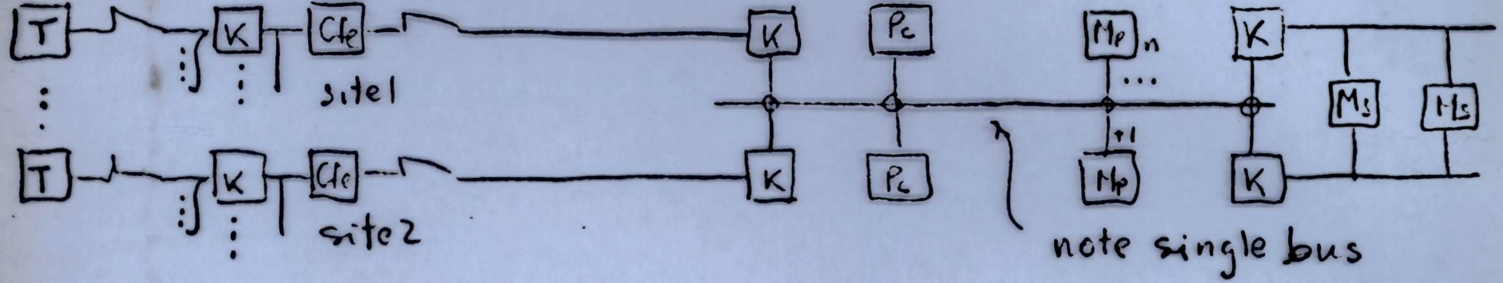


Fig 17 N - 2P_c (1 Bus) structure, 1~3 sites

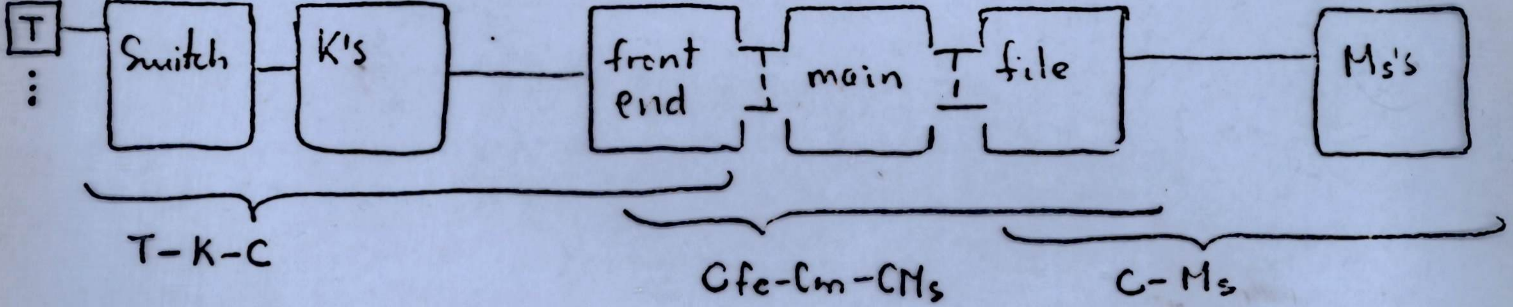


Fig 18. Problem interconnection / structure.

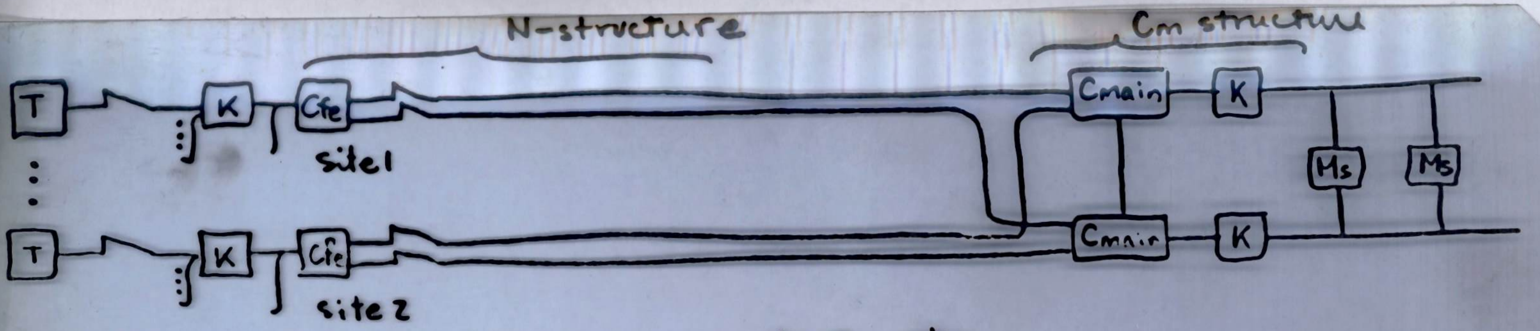


Fig 12 N-Cm structure 1~3 sites

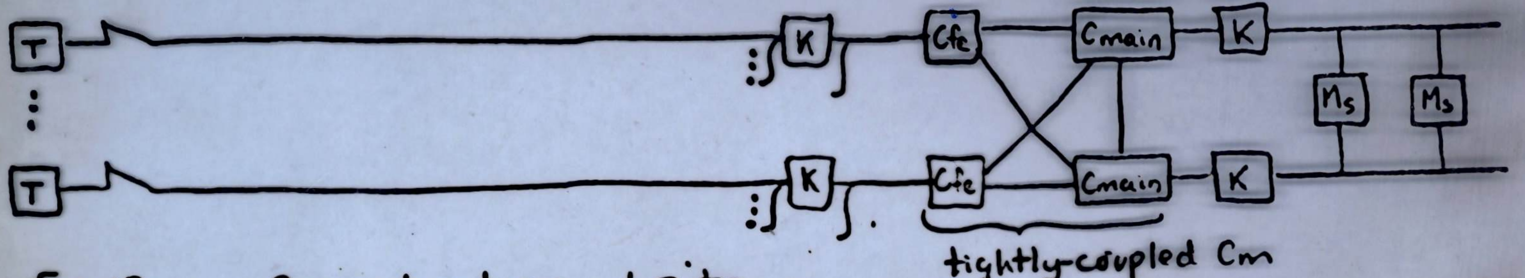
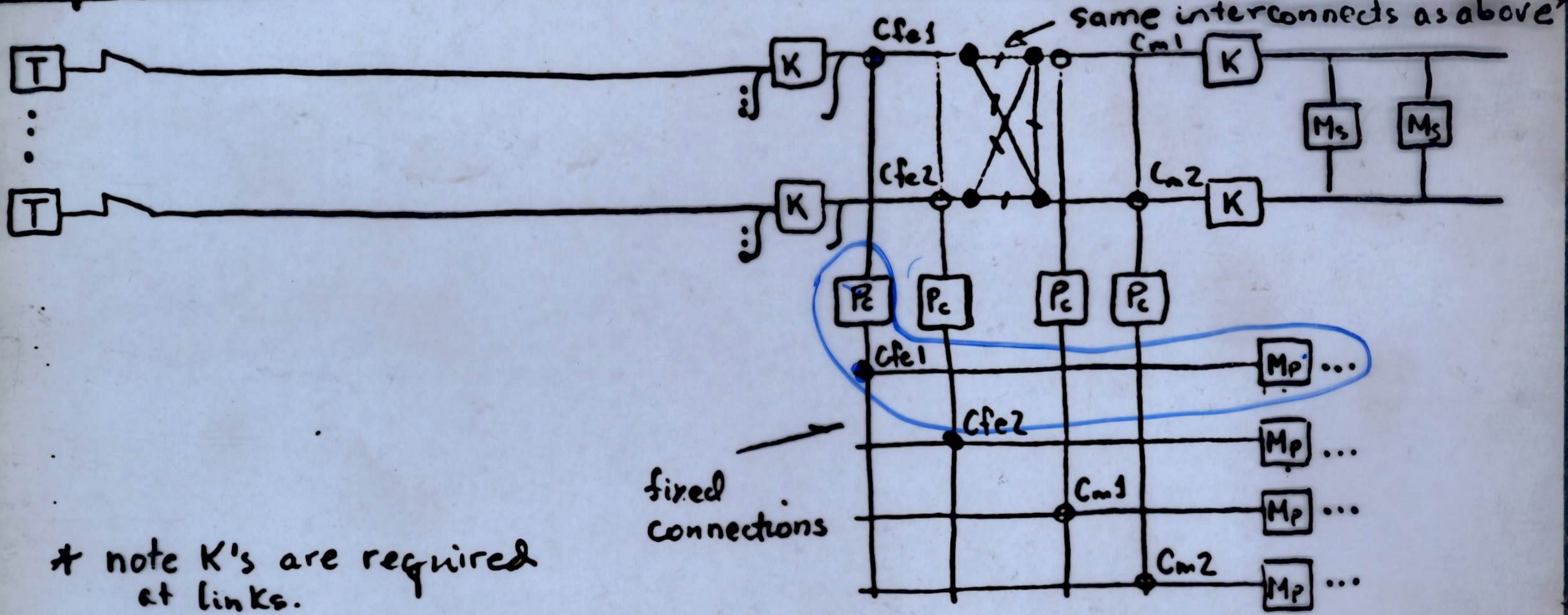
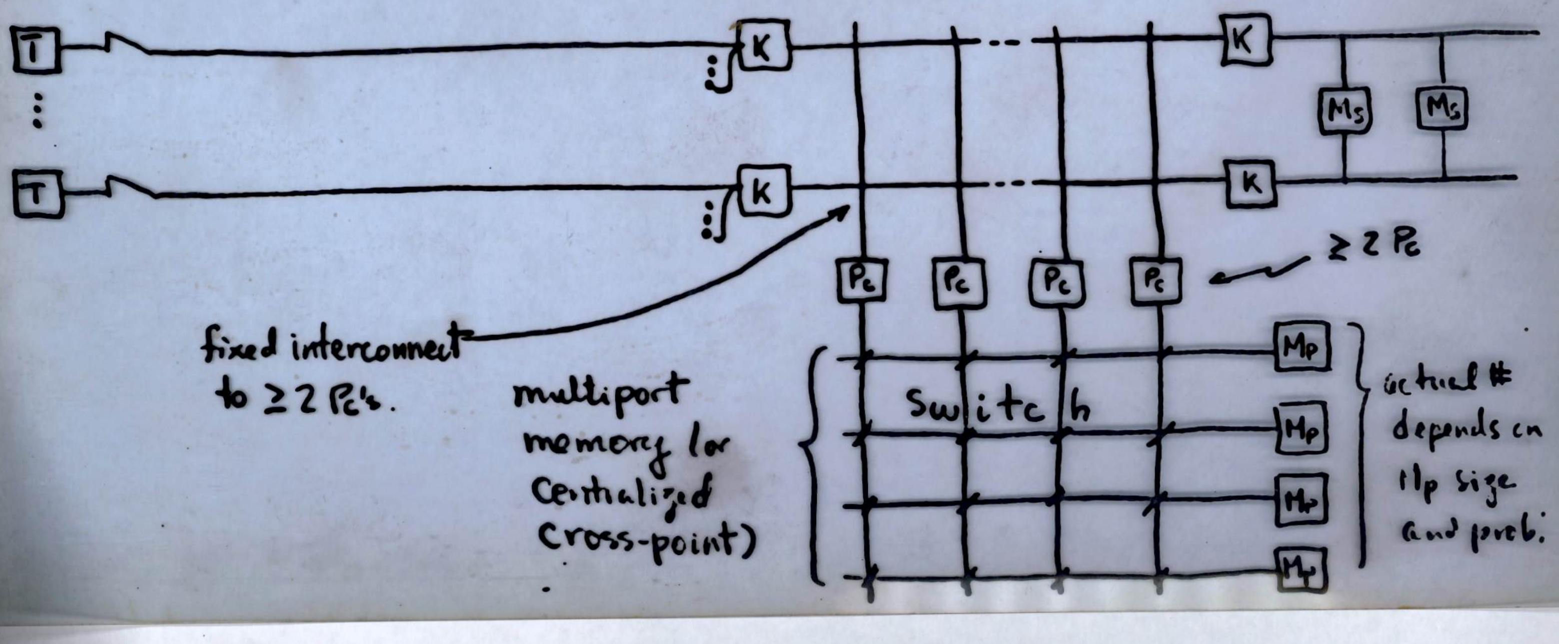


Fig 13 Cm structure, 1 site



* note K's are required at links.

Fig 14 Cm structure, 1 site (Fig 13 redrawn as a potential multiprocessor


T
K
Ms
Pc
≥ 2 Pc
fixed interconnect to ≥ 2 Pc's.
multiport memory (or Centralized cross-point)
Switch
Mp
actual # depends on Mp size and prob.

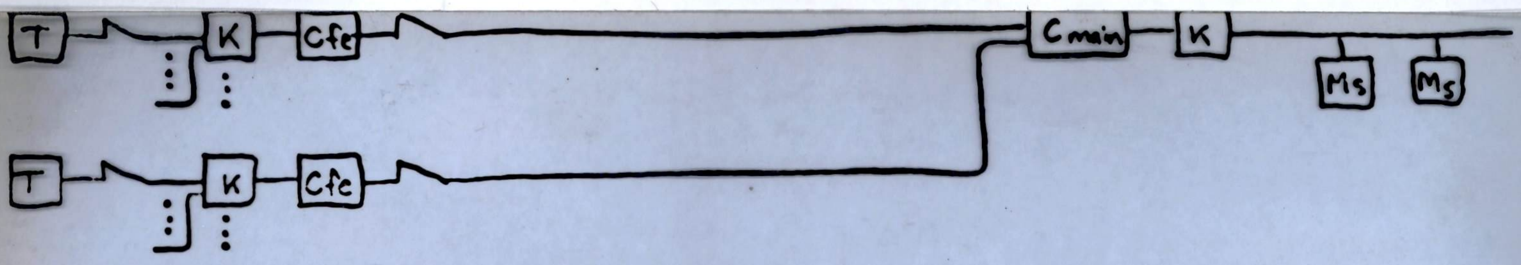


Fig. 5 N := (2 Cfe, at 2 local sites + Cmain), 1~3 sites

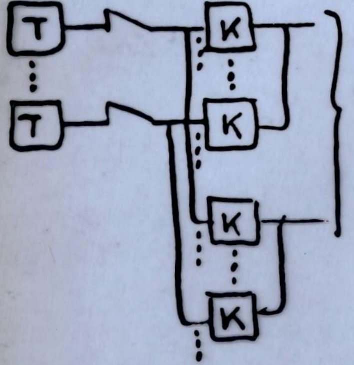
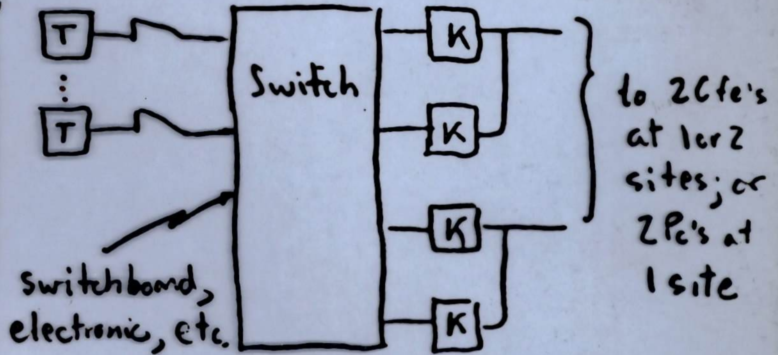


Fig. 6 Bussed T's connected to redundant K.comm's, 1 site



T
T
Switch
K
K
K
K
switchboard, electronic, etc.
to 2 Cfe's at 1 or 2 sites; or 2 Pc's at 1 site

Fig. 7 Duplex switch to redundant K.comm's, 1 or 2 sites

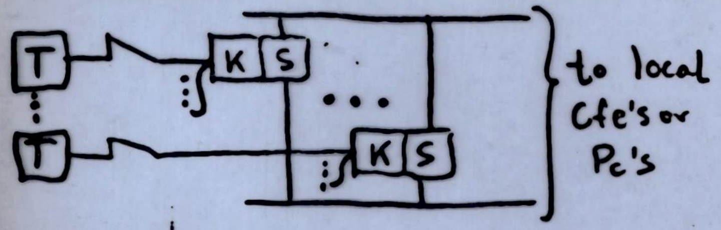


Fig. 9 Duplex switch at each K.comm to redundant Cfe's or Pc's, 1 site

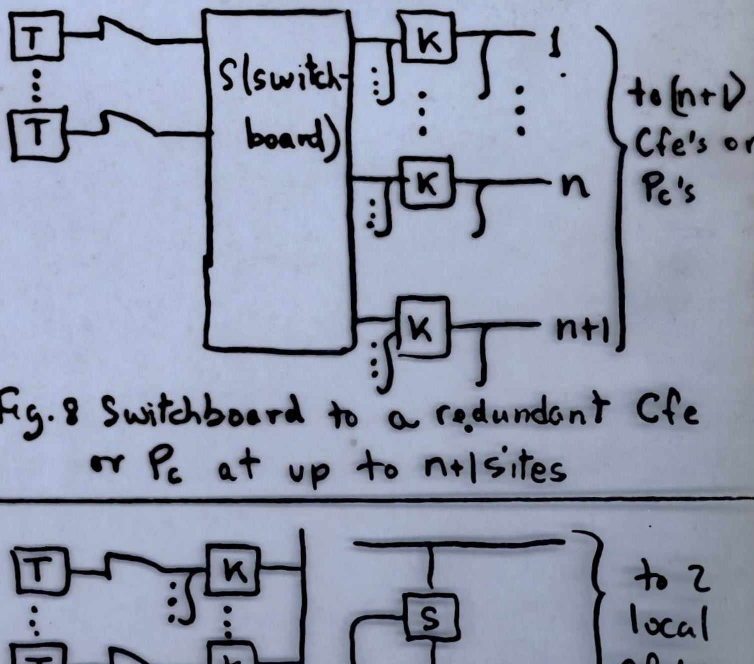


Fig. 8 Switchboard to a redundant Cfe or Pc at up to n+1 sites

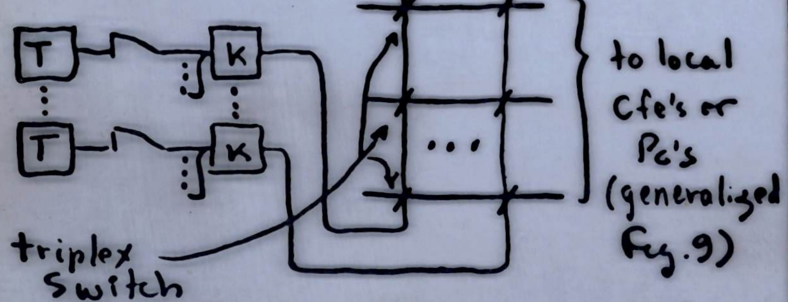


Fig 10. Triplex Switch at each K.comm to 3 (Cfe's or Pc's) at 1 site

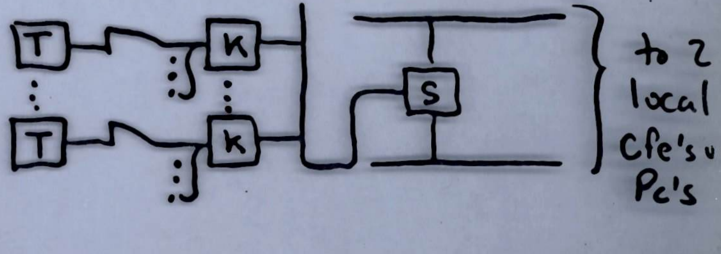
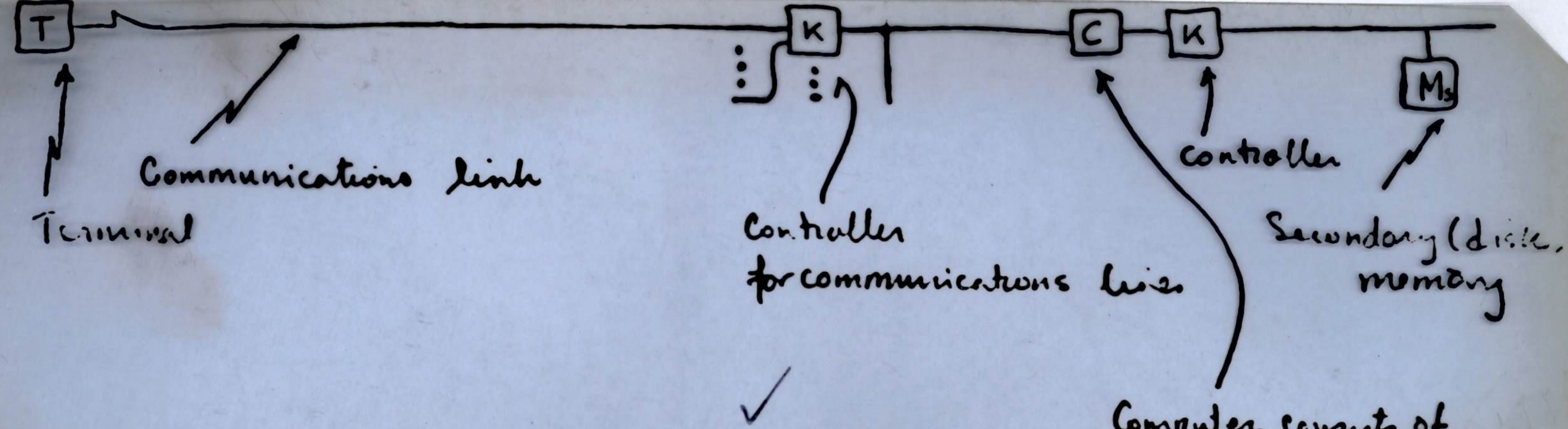


Fig 11. Unibus switch for single K.comm's to redundant Cfe's


Terminal
Communications link
Controller for communications lines
Computer consists of Processor (Pc) + Primary Memory
controller
Secondary (disk) memory
T
K
C
K
Ms

✓

Fig. 1 Basic C, 1 site

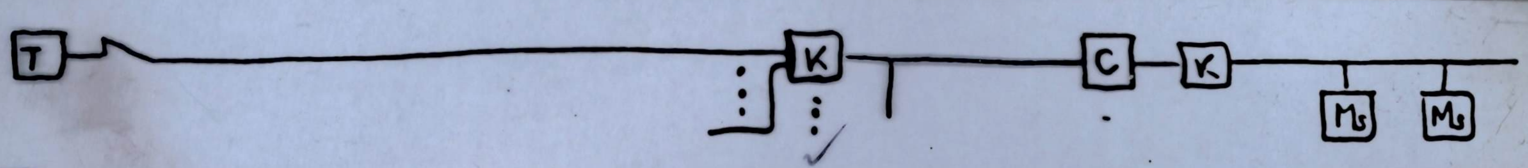


Fig. 2 Basic C with redundant disk, 1 site

Fig. 2 Basic C with redundant disk, 1 site

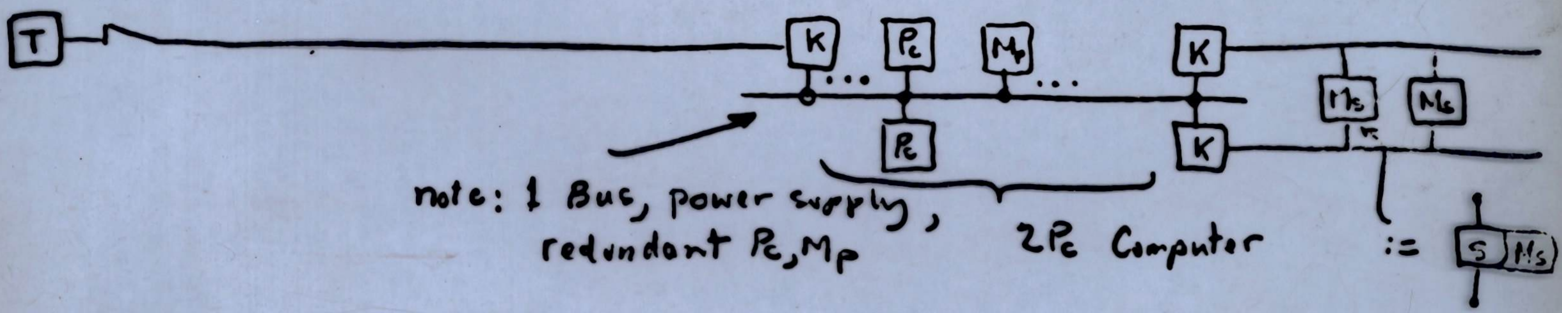
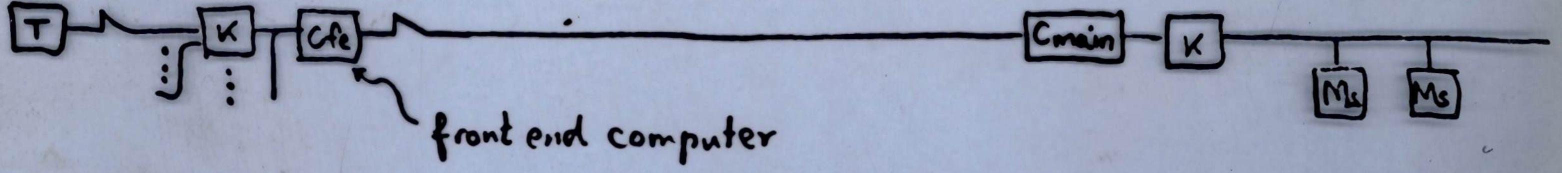


Fig. 3 2 Pc, 1 bus C, 1 site

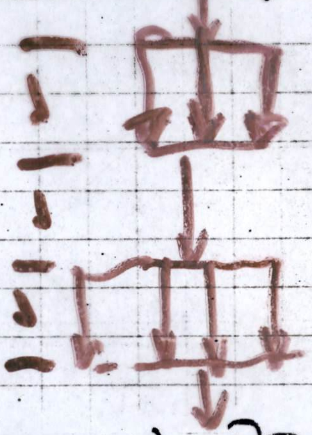


PARALLELISM: BASIS FOR MULTI-P's

Intra-C Parallelism (IO BOXS, F/E B/E) Daemons,

STATIC (I.E. RELIABLE)

- INDEPENDENT PARTITIONS: — DYNAMIC
  - Pi: STOCKROOM. — Space sharing.
- FUNCTIONAL SEPERATION — FRONT/BACK END / PRE-PROC. /



Grain-size

- INDEPENDENT Tasks.
  - i.e. multiprogramming
  - JOBS (T/S)
  - PROCESSES (TRANSACTION PROC.)
  - BATCH STREAMS
- SET PARALLELISM (PROC. CONT)

small.

- ARRAYS, VECTORS
- GENERALLY CONCURRENT PROCESSES.
- PIPELINE

Intra-task.

→ multi-Processors (tightly
→ Computer module Coupled)

Networks

STRUCTURES

1. MULTI-PROCESSOR - mP
2. COMPUTER MODULES - Cm
3. NETWORKS.

PROBLEM / OBJECTIVE FCN

1. LOCATION OF "WORK"; PEOPLE
2. REQ. FOR P_c, M_p, M_s, AVAILABILITY

TRADEOFFS

	+	− (and COST)
mP	LESS M_p, L, MORE P_c AVAIL; CONFG. TO MAINTAIN	SM_p; INTEGRAL SYSTEM.
Cm	FUNCTIONAL ISOLATION	M_p, Links, P_c (ABILITY TO MATCH/MOVE WORK.
N	LESS FCN. ISOLATE. COMM. LINKS.	SAME AS Cm ~~(LINKS MORE)~~ less able to move wk.

	Network	C-Modules	Multi Proc.
proximity	long dist.	← 1 room	→
Datarate	10~50 Khz	1~10 Mhz	→
Response.t	> .1 sec	≈ 1 ms	→
Port Arch.	Synch. Com.	Shared address + interrupt	Shared Mp.
Applications	Distributed Computing, Terminals.	High perf.	→
		Fixed Applic.	General purpose Applic.
	Load share, File ″ ,	Special by Function	Perf Δperf. Rel. in
Ability to grow.	Rejuvination	Fixed	→ performance varies with (t, $)

copy to Dan + ret.

✓

Gordon Bell
December 4, 1975

MULTIPROCESSOR, MULTICOMPUTER AND COMPUTER NETWORK STRUCTURES

COMPANY CONFIDENTIAL

Background and Motivation

We (DEC, our customers, and the computer engineering community generally) have been building various multiprocessor and multicomputer structures for the last few years. The DECNET structure was derived from this base. It is a propitious time to focus on the more conventional multiprocessor and multicomputer (tightly coupled computer structure) because:

0. Users are becoming aware of their existence, are asking about them, and our marketing groups are beginning to drive this way.

1. We are selling tightly coupled systems as minicomputers, and their design is clearly not well understood. We need various techniques to engage in these architectural designs. Two design styles: functional multiprocessing and duplex computers are used here providing poor availability and cost/performance, respectively. Multiprocessors could solve these basic problems better.

2. The number will increase as the LSI density increases, and places more emphasis on production, rather than design, as a way to obtain performance. Multiprocessors are a better way to utilize LSI than unique, complex designs. I believe the large VAX machine and the smallest chip 11 are the last machines we will ever build that are not explicitly multiprocessor oriented.

3. They represent the only way to achieve arbitrarily high availability and performance.

4. The systems can provide better characteristics than a uniprocessor in terms of:

   A. More reliability.

   B. More availability

   C. Greater performance, and better cost/performance characteristics.

   D. Incremental field expandable performance increase along the processor dimension.

E. Less basic designs to get a wider performance range. This also gives us and our customers less parts to stock.

5. We have been applying the PDP-10 as a 2 processor, multiprocessor for the last 5+ years effectively.

6. We have several research and advanced development efforts going now, and would like to begin to communicate the results and check their applicability.

7. According to the older Telex papers, IBM is to move to these structures in the 1976 time frame, with what was then called FS (for Future Systems). There is mixed review about whether this will happen..my guess is it will. Thus it will then not be a matter of being nice to have, but rather market demands.

8. The tightly coupled structures provide for graceful rejuvination of their computers. With this, a mini front end is first added to an "old dog" to off load the large beast. Second, files are moved to the front end. Third, the application is now resident in the "front end". And finally, the "old dog" is removed.

TABLE OF PROS AND CONS FOR BUILDING MULTIPROCESSORS

Pros	Cons.
0. General market appeal	IBM's yet to bless concept. Education needed before they can be fully utilized
1. Greater availability	More care is needed in
design through multiple (redundant) components	
2. User may configure a trading	Can do this by renting and
system with right	in model n for n + 1.
processing capacity (i.e.	system grows with his load).
3. Arbitrarily high loss	May not totally materialize;
performance interface.	of performance through
4. Highly cost-effective single	Programming dependent for
for:	tasks (i.e. no easy way to
-multiple-process/ into	automatically break a task
multi-tasks as in RSX-	sub-tasks for parallel
execution) type, IAS, and transaction processing.	
-multi-programmed/time shared as in RSTS and 10-op.	
sys. -multiple function systems as in front end and	

file-type processors, -explicit parallel programming; and concurrent programming

5. We design, produce, We may not be able to match specific
stock, and sell fewer market niches so precisely
types (resulting in lower prices)

6. Higher availability Faulty component hard to find and
through multiple may propagate errors.
(redundant) components

7. Voting designs possible Loss of performance; explicit
for extreme reliability programming may be required.

8. Technology is making them the "best" way to design a computer

Why Multiprocessors Have Not Existed

Although it is not surprising that multi-processors have not been used except on a highly specialized basis, it is depressing. In Computer Structures (Bell and Newell, 71) we carried out an analysis of the IBM 360, predicting a multi-processor design. The range of performance covered by the PDP-11 models is substantially less than with the 360, although the competitive environment of the two companies is substantially different. For the 360, smaller models appear to perform worse than technology would predict.

The reasons why multiprocessors have not materialized may be:

0. The set of seven arguments put forth in a paper by Bill Wulf and I (Wulf & Bell, 1972); the paper is included. Most of these arguments have been overcome. They are:

	Reason	Reply
1.	high cost of Mp and Pc	Use minicomputers
2.	relatively high cost of Pc negates effect of incremental improvement	Pc costs are tending to be negligible part of the system
3.	unreliability of complex software	we understand this now; furthermore systems are structured to support multiprocessors
4.	inability of switch technology	all right now
5.	memory conflict thought to be high	now understood; and is not high for right balance of Mp + Pc
6.	unknown problems of dividing tasks into sub-tasks	much work on parallelism; however, systems for real time, time sharing

	and transaction processing are inherently parallel by task, and/or program. (We don't need to divide below task level).
7. probelms (mechanisms) for parallel environment	these already exist for all multi-tasking operating systems

1. The basic nature of engineering is to be conservative. Given there are a number of risks in a product already, it is unclear why one should build a structure that may require a new way of programming (with a higher project risk). This is a classical deadlock situation: we cannot learn how to program multiprocessors until such systems exist; a system cannot be built before programs are ready. One has to believe that the benefits are great enough even without extensive reprogramming to merit this structure. Fortunately, process-based operating systems are oriented to multiprocessor structures.

2. The market doesn't demand them. Another deadlock: how can the market demand them, since the market doesn't even know that such a structure could exist? Although multiprocessors are used extensively in larger systems by Burroughs and Univac, IBM has not yet blessed the concept.

3. We can always build a better single, special processor. This design philosophy stems from local optimization of the designed object, and ignores global costs of spares, training, reliability and the ability of the user to dynamically adjust a configuration to his load. In all dimensions of computer space, there is dynamic variability: Mp-size, Ms-size, and number of terminals. Pc performance could be continuously variable in the same way.

4. There are more available designs for new processors than we can build already. Within our environment, the computer design group has a great deal of power and status. Given this situation, there is little reason to have fewer products (and fewer groups working on better products).

5. Planning and technology are asynchronous. Within DEC, not all products are planned and built at a particular time, hence, it is difficult to get the one right time when a multiprocessor would be better than an existing uniprocessor together with one or two additional new processors. New technology also makes the process difficult by providing opportunities at asynchronous times.

6. Incremental market demands require specific new machines. By having more products, a company can better track competitors by specific uniprocessors.

7. Quite possibly they are the wrong way to build computers. There is a chance this is true, but I don't believe it.

Brief Introduction to, and Definition of the Structures

As computer engineers, we have built and are building various kinds of multi-processor computer systems. Rarely do we see a computer structure that is not interconnected to another computer in some way. The systems are most easily characterized (differentiated) by the degree of interconnection among the various parts, i.e. the way the parts are linked together both in terms of the hardware and software. Although, this characterization appears fuzzy, and there is some overlap, the following definitions are reasonably widely accepted. The three, rather clear, PMS structure architectures* are:

Symmetrical multi-processor/mP**.

This computer consists of more than one central processor, Pc, which share a (large) primary memory, Mp. That is, any Pc can execute the program (or part of it) within Mp. Each Pc, may also have some private Mp for performance or reliability reasons. Within this broad category of physical, PMS structure architectures, the use (programming) can take on a range of forms, which are also characterized, by the degree of coupling among the program parts. Some of the interesting points in the range are:

parallel processing - where all processors execute a single task in an array, pipelined, or concurrent fashion;
through multi-programming - where each processor operates on a single program at a time (with the implication that multiple programs are available to work on as in multi-stream batch, timesharing, and transaction processing systems);

on to functionally separated processors;

and finally independent (segmented) computers - where each Pc is artitioned with its own part of Mp and peripherals.

Tightly-coupled computers\multi-computers\mC\Computers module\Cm.

This is fundamentally a network of computers which are usually located within a short distance of one another at a single site, and interconnected via high speed links (e.g.

>1M bits/sec). Work is usually divided up among the computers on some sort of functional basis. The most prevalent form of Cm is a C.duplex where two computer operate together on a single problem with one being a backup.

Computer.Network\Network\N..

This is a collection of Computers\C's, interconnected by communication links\L's which are relatively low speed (i.e. 5K to 50K bits/sec). The computers are usually not at the same physical site.

Thus, the definitions can become fuzzy and is a continuum:

```
Symmetrical multi-Pc    Computer Modules                Networks
      Assymetrical      (tightly coupled)
|-----|------------------------|--------------------------------|

|<- direct memory access------------->    <-----serial links---->

                        <- high speed links -->

<--------------------- program in a C.module-->
                        may access data and/or
                        execute program in another module.

                        The above continuum has ignored the
                        very tightly coupled aerospace-type
                        and telephony double and triple
                        redundant computers which use voters
                        to decide, after each instruction,
                        what the correct result is.
```

The opportunities to build various structures will vary over time with the technology, and Cm structures may look more like mP structures as very high speed link costs decrease. On one hand, as Cm's are interconnected by very high speed links, with capability to access one another's Mp, they appear to be more like mP structures. On the other hand, as Cm's are built with lower speed links, and communicate more on a message basis, they appear to be identical to N's. Digital's DECNET protocols handle N's and Cm's in a transparent fashion, by permitting any speed links to be used among networked computers. In the paper, we will show by simple transformations how to move from structure to structure. A structure will very often be a hybrid collection of the above techniques not purely one or another.

*PMS Structure Architecture denotes the physical interconnection (structure) of computer components. The component types are: P** processor; M\memory; S\switch; K\control; L\link; T\transducer (also terminal); D\data- operation unit; C\computer; and N\network of C's. We use the PMS notation to denote and describe both the hardware and software architecture structures. For those unfamiliar, an appendix is included to fully introduce PMS in some detail, or one can refer to several books and papers. However, for the majority of readers who don't want to be bothered, the definitions are given as needed in the text.

**The symbol \ is used for defining synonyms in passing.

Observe the following analytical and experimental results for using multiple 11/05's versus an 11/40. Similar arguments could get us to the 11/45.

Pc's	Rel.	=Rel. Price		Relative Price/Performance	
(11005's).	Perf...	Sm.System...	Lg.System..	Sm..System...	Lg..System.
1	1	1	3	1	1
2	1.85	1.23	3.23	.66	.58
3	2.4	1.47	3.47	.61	.48
11/40	2.25	1.35	3.35	.6	.49

Namely, putting two 11/05's on a single bus has a dramatic effect on performance <u>and</u> cost-effectiveness. Up to three 11/05's will still increase performance and cost-effectiveness. Four processors begin to saturate the bus and memory (here about .7 us), and little additional performance increase is obtained and the cost-effectiveness begins to increase again.

Using this method, we could have obtained 11/40 and 11/45 performance with a single design since the dynamic performance range from 11/05-11/45 is only about 5 (ignoring floating point).

The point is: we should move into multiprocessors instead of re-engineering processors to cover a range, particularly since the performances are so close among the machines.