CMU-CS-78-143

Reliability in Multiprocessor Systems: A Case Study of C.mmp, Cm* and C.vmp.

Daniel P. Siewiorek, Vittal Kini (Editors)

Contributors:

Part I: Vittal Kini, Henry Mashburn, Stephen McConnel, Daniel P. Siewiorek, Michael M. Tsao.

Part II: Harold Bellis, Rostam Joobbani, Vittal Kini, Daniel P. Siewiorek.

ABSTRACT

This report consists of two papers that treat reliability of the multiprocessor systems at CMU. The first paper discusses the multiprocessor architectures, reliability features (hardware and software), and measured reliability data. The second paper presents hard failure data from one of the systems, calibrates a hard failure rate model, and analytically models the reliability of the three systems. These papers will appear in the October 1978 issue of the Proceedings of the IEEE (Special Issue on Fault Tolerant Systems).

A Case Study of C.mmp, Cm*, and C.vmp --
I. Experiences with Fault Tolerance in Multiprocessor Systems

Daniel P. Siewiorek, Vittal Kini, Henry Mashburn,

Stephen McConnel, Michael Tsao

Departments of Computer Science
and
Electrical Engineering
Carnegie-Mellon University
Pittsburgh, Pa.

Abstract

Three multiprocessor systems designed, implemented and currently operational at Carnegie-Mellon University are compared and contrasted. The design goals and architectures are summarized with a special focus on reliability features. Experiences gained in design and operation are discussed. Finally, reliability data, with a focus on transient failures, measured from each system is presented and discussed.

Keywords
Multiprocessor, Transient failure, Reliability, Fault Tolerance.

This work was supported in part by the Advanced Research Projects Agency under contract number F44620-73-C-0074, which is monitored by the Air Force Office of Scientific Research; in part by the Office of Naval Research under contract number NR-048-645; in part by the National Science Foundation Grant GJ 32758X; and in part by Digital Equipment Corporation.

Table of Contents

1. Introduction

In 1970 Carnegie-Mellon University initiated a design study into computer structures supporting a high processor-to-memory bandwidth. Since that time three multiprocessor systems have been designed, implemented, and made operational. While all three systems had differing goals and technological constraints, they shared one fundamental design decision: use as much commercially available hardware as possible. Several advantages resulted:

1) Limited design resources could be focused on the interconnection architecture rather than being diluted doing state of the art processor, memory, and I/O fabrication.

2) Existing products had a library of software available, especially diagnostics.

3) Commercial modules enjoy increased reliability due to volume production. As illustrated in a companion paper [1], this could mean as much as a factor of ten increase in mean time to failure.

4) Parallel programs and support facilities could be shared between the multiprocessors if they shared the same instruction set.

The three following sections on the architectures adhere to a common format. First the design goals of the system are sketched, followed by a brief discussion of the architecture including the reliability features. Experience gained with the reliability of each architecture is followed by a presentation of actual data measured from the system. The data focuses on transient failures and their causes. Each section concludes with an indication of future research on the architecture.

The three architectures are C.mmp (a multi-minicomputer), Cm* (a modular multi-microprocessor), and C.vmp (a voted multiprocessor). Design concepts for C.mmp were initiated in 1970 at a time when a minicomputer cost $10,000. The goal was to establish a high performance, low cost multiprocessor for work in speech and image understanding. Because of the relatively high cost of hardware, a spartan architecture was implemented, leaving reliability and resource management to software. Advantages and limitations of the software approach to reliability are contained in Section 2. Initial design studies for a follow-on multiprocessor, Cm*, that took advantage of LSI technology was started in 1972. Potentially a 100 processor system was envisioned, but final design work had to wait for the advent of the $1000 LSI-11 microcomputer in 1975. Building on the C.mmp experience, a substantial portion of the

Cm* design was devoted to reliability and operating system support as outlined in Section 3. In 1975 a design study was started for a low end processor that could tolerate transient faults anticipated in process-control applications. Other goals included transparency to the user of error recovery and on-line maintenance without loss of computing power. Section 4 illustrates the transient and hard fault survival capacity of C.vmp as well as the performance degradation under faulty conditions.

2. C.mmp, A Multi-Miniprocessor

2.1 C.mmp Architecture Overview

C.mmp is the oldest of the three multiprocessors at CMU. First we will present the current PMS (Procesor, Memory, and Switch) [2] structure followed by a discussion of the CMU-built hardware components that have most affected reliability.

2.1.1 The PMS Structure

C.mmp is composed of slightly modified DEC PDP-11/20 and PDP-11/40 processors. As shown in Figure 1, up to 16 of these processors, in any mix, may be connected to 16 ports of shared memory by a 16 X 16 crosspoint switch (Smp), providing an address space of 32 megabytes [3]. The basic modifications to the processors were: to make user execution of certain privileged instructions illegal (e.g. HALT, RESET, WAIT, RTI (ReTurn from Interrupt) and RTT (ReTurn from Trap)), and address bounds checking on the stack pointer register, R6. These modifications were required for software protection. The operating system must leave some context information on the stack over protected procedure calls [4]. RTI and RTT were modified since they modify the processor status (PS) word and it must be protected because it controls the memory protection scheme (see section 2.1.2). However, these features have also had significant impact on hardware reliability. The 11/40's have been additionally modified to allow an extended, writable control store [5].

For about the last year, the configuration has consisted of 5 11/20's and 11 11/40's. Recently, the non-microprogrammed 11/20's have been dropped from the configuration to allow greater use of special instructions within the operating system. All data in this paper refer to the full 16 processor system with 2.5 megabytes of shared primary memory.

An interprocessor bus connects the entire set of processors. This bus provides three basic funtions:

1. interprocessor interrupts at three priority levels,

2. the control functions halt, continue and start,

3. continuous broadcast of a 60 bit non-repeating clock value used for interval timing and unique name generation in the operating system (internal operations of the individual processors are not synchronized by this external clock).

The relationships of the processors, memory and bus are shown in Figure 1. Also shown are the per-processor 8K byte local memories (Mlocal) and the principal secondary memories Mdisk (RP02 and RP03 2314-type disks) and Mpaging (one megabyte fixed head disks with zero latency controllers for paging space). Note that peripheral devices are assigned to the UNIBUSes of specific processors. I/O requests are mapped from requesting processors to the processor controlling the device via an interprocessor interrupt.

2.1.2 Shared Memory Access

Access to shared memory is performed in two stages: relocation of the 18 bit procesor-generated address into a 25 bit address space, and resolution of contention in accessing that memory location. The relocation units (Dmap) divide the 32k-word space into eight 4k-word pages which may be anywhere in shared memory. There are four address

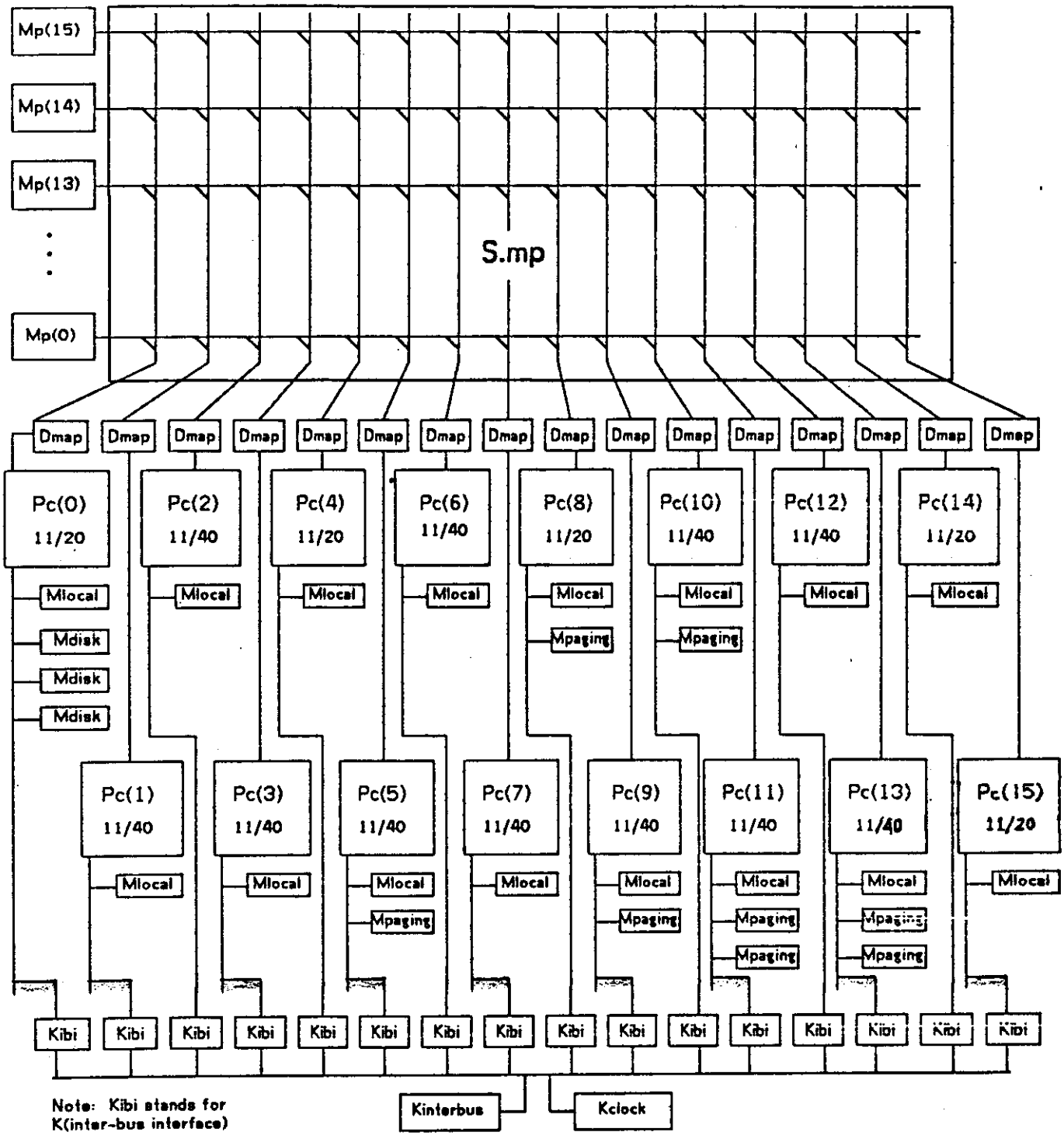


Figure 1 The PMS Structure of C.mmp

spaces, specified by two bits in the PS. Therefore, four sets of eight registers are provided in each relocation unit, although the stack page is common to all spaces to allow communication across spaces.

The four address spaces are the heart of the memory protection mechanism: in only one space (1,1 in the PS space bits) are the relocation registers directly addressable. Since this space is used exclusively by the Hydra kernel [4], protecting the PS guarantees that no addressability changes may be made without the approval of the operating system. All entries to the kernel, whether by interrupt or user request, force the assertion of both space bits. Direct addressability is accomplished by disabling two of the relocation registers in (1,1) space, one each for Mlocal and the control register bank for all peripheral devices (including Dmap). With these registers disabled, addresses that would normally be mapped are passed along the UNIBUS unchanged to be received by the addressed register or memory location.

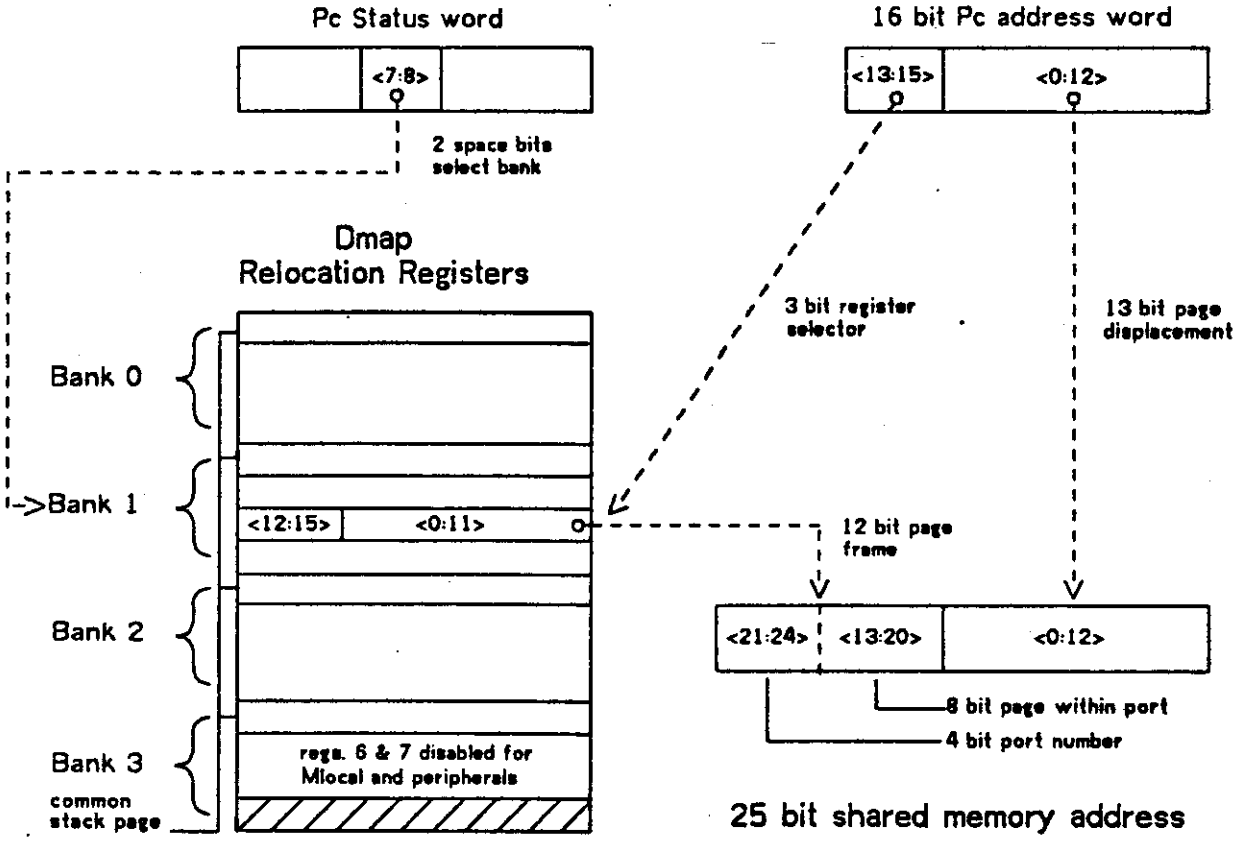


Figure 2 C.mmp Address Relocation

As illustrated in Figure 2, the relocation unit intercepts the 18 bit UNIBUS addresses (16

bit words plus the two space bits) and converts them in the following manner: the three high order bits of the 16 bit word select a register from the bank specified by the space bits. The contents of the register provide a 12 bit page frame number; the remaining 13 bits from the address word are the displacement within that page. The two are concatenated to form the 25 bit shared memory address. This transparent mapping is performed for all shared memory accesses. In addition to the 12 page frame bits, there are four bits in each relocation register used for control. They are designated: "no page loaded", "write-protected", "written-into", and a bit to control whether values from the page may be stored in the planned, but not yet implemented, per-processor cache.

The shared memory address and possibly 16 bits of data, each parity checked, and two bits of access function data are sent to the crosspoint switch. The address parity is checked at the switch interface. If the check fails, the request is aborted and the processor interrupted. Data parity is not checked until the data is read from memory. All parity is generated, and data parity checked, by the relocation unit (Dmap) interface to the bus from the switch.

The switch then routes the request to the memory port specified by the high order four bits of the address. A port is requested by setting the processor's bit in an initial register, the request register. Contention for the port is resolved by periodically gating the request register into a second register, the queue register, which is left-shifted as the port becomes available. The shifting creates a priority ordered queue: as a bit is shifted out, the corresponding processor is granted access to the port. Processor 15 is assigned the high order bit; processor 0 the low order bit, defining the priority. When the queue register is zero, all requests have been satisfied. The request register is again gated into the queue register, cleared, and a new cycle begins. A second request for the same port by a processor must enter via the request register, hence equality of service among the processors is maintained. This two level request mechanism also obscures the internal queue's priority ordering to the point that it is of virtually no importance outside of the switch, preserving the symmetrical design of the crosspoint. The switch's maximum concurrency (16 independent paths) is achieved if all processors request different ports.

The cost of address translation, switch overhead (no contention), and roundtrip cable overhead is about one microsecond. Although this is high by today's standards, more than equal to the access time of the memory, it has not proven prohibitive.

2.1.3 The Interprocessor Bus

The interprocessor bus provides a common clock as well as interprocessor control. These two logically and functionally separate features travel separate data paths although they share a common control (Kinter-bus). Each processor has an inter-bus interface (Kibi) that defines the processor's bus address and makes available the bus functions to the software.

The first function is to continuously broadcast the 60 bit, 250KHz Mclock. This is done by multiplexing the clock value onto a 16 bit wide data path in four time periods, low order bits first. Any Kibi requesting a clock read waits for the initial time period and then buffers the four transmissions in four local holding registers available to the software. Clock values are often used for unique names [4],[6] so the otherwise unused high order four bits of the fourth local register are set to the processor number (bus address) to insure uniqueness when any number of Kibi's read the bus simultaneously.

A count-down register is also maintained in each Kibi for interval timing. It may be initailized to a non-zero value by the program; 1 is subtracted every 16 microseconds (timing supplied by Mclock) and the processor is interrupted when the register reaches zero.

The second bus function is the interprocessor interrupt and control mechanism. Each processor may interrupt, halt, continue, or start any other processor, including itself. The control operations are invoked by setting the bit(s) corresponding to the processor(s) to be controlled in a 16 bit register provided by the Kibi for the desired operation. A second 16 bit wide data path is eight-way time multiplexed. Each control operation is assigned a time period. As the appropriate period arrives, each Kibi OR's its control operation register onto the bus and clears the register. Synchronization of bus access, as well as operation specification, is accomplished by the multiplexed time periods. The Kibi also inspects the bus to see if the specified operation is being invoked on its processor; if so, the requested action is performed. Although eight time periods are available, only six are used: three priority levels of interprocessor interrupt, halt, continue, and start; the remaining two are ignored.

2.2 Fault Tolerant Mechanisms in C.mmp

Having some knowledge of the workings of C.mmp, its fault tolerant mechanisms can now be examined. The emphasis will be on the reliability of the fault tolerant mechanisms, but other areas such as the memory modules, hardware features to enhance software reliability, and software failure recovery models have all had great impact on the total system reliability. An exhaustive treatment, especially in the area of failure types, is not intended. (see also [7])

2.2.1 Hardware Mechanisms for Fault Tolerance

The necessity of constructing C.mmp from available minicomputers greatly restricted the possible fault tolerant mechanisms that could be incorporated. For example, neither of the two PDP-11 models used, nor the UNIBUS, have error checking abilities; one must assume that their results are correct. Experience has shown that this is not always the case... Therefore elaborate error checking and correcting of the shared memory and its access path were not justified; only simple parity checks are done. Even so, there is room for some cleverness: since there is a separate parity bit for both bytes of the word, one byte is given odd parity, the other, even. This detects words of all 1's or all 0's, both of which are common results of transient timing failures.

The interprocessor bus has no error checking whatever. All checking and failure recovery are done by software, which has been highly successful in this case.

The most elaborate checking is done to ensure software integrity. The stack pointer (SP) is required to be within the stack page and have an even (i.e. word address) value. With hardware protection enabled, it is impossible to load the SP with any other values. The SP is further restricted to lie between a fixed overflow limit and a variable underflow limit. This is necessary to protect operating system information that must be left on the stack during user execution. Due to the difficulty of modifying the processors, the stack underflow register (SUR) and the comparison circuitry were physically placed on one of the relocation unit boards. This remote placement compounded the timing difficulties of adding stack limit checking to the processors. Having to protect the PS by disallowing user execution of RTI and RTT increased the perturbation of stack operation timing. Unfortunately both of these modifications were necessary to insure safe operation of a multiprocessing, multi-user operating system.

2.2.2 Reliability Experience

In spite of the above mentioned difficulties, the machine has been reasonably reliable,

considering its highly experimental and unique nature. Recent statistics indicate that the total system MTTC (Mean Time To Crash), counting all forms of errors that produce a crash, is, with one exception, fluctuating between 6 and 15 hours, averaged on a monthly basis. This is more than enough uptime to be a useful computing engine, especially since the average downtime after a crash is only about five minutes and the machine automatically reloads itself (operator intervention is virtually never required).

The remainder of this section is a retrospective view of the principal failure types encountered on the project. Many failures that were once common are now rare or non-existent, others are still apparent and some reappear from time to time. The failure rate has been significantly improved over the last year through a program of intensive maintainence. This program has been in progress since completion of the basic 16 processor machine.

Memory parity failures have, with rare exception, been the most common falure mode. Most are transient, but hard errors happen with regularity. Often the memory failure rate has largely determined the MTTC. A methodology for recovering from transient memory failures in the shared memory of the operating system is now being developed. A marked improvement in reliability is expected from this one recovery effort. (Most memory parity failures happen in the operating system kernel. The kernel executes shared code with a high degree of parallelism, resulting in memory port contention. This is thought to be the cause of the failures appearing in the kernel. Memory areas in user-allocated pages present a lesser problem. The recovery methods for both areas are presented in the next section.)

Transient failures, while it is always difficult to isolate their source, have been an especially large problem on C.mmp since there are few, if any, trace points in most data paths. Not including powerful debugging aids in the logical design has continuously hampered development. There was little that could be done for the processors, but aids could have been incorporated in all the CMU-built logic. Realizing this weakness, one tracking register (for the program counter) has been added to the relocation units and another (for operand addresses) is being developed. A similar weakness became evident in the software: often information about a failure was lost by the operating system, making recording of the conditions for transients unreliable. The latest rewrite of the crash logging procedures has alleviated this to a great extent.

A transient failure that has eluded solution is the problem of "false NXM's". The processor reports a non-existent memory (NXM) exception, but upon analysis, the memory is responding, and the instruction, registers and index word(s) are well-formed. No exception should have resulted. Intermittent timing problems are suspected, but there is insufficient information available to isolate what may be failing.

Another long-standing transient is stack operation problems. This usually appears as misexecution of subroutine call/return instructions or interrupt entry/exit mistakes. The most common form of the error is one too many (few) words pushed (popped) from the stack. This failure is thought to be a side effect of the SP checking modifications and disallowing RTI and RTT, but the cause has never been isolated. The transient is relatively rare and no method of recovering from it has been developed.

A pleasant surprise has been the reliability of the crosspoint switch. Although it is the most complex component of the multiprocessor hardware, it is now among the most reliable. No doubt the relatively simple design, conservative implementation, and careful construction have paid off. However, an early problem required considerable effort to fix. Certain conditions, characterized by a memory access not completed by the UNIBUS master, could cause the switch to deadlock due to the lack of a time-out circuit in the memory port control

logic. Any other processor attempting to access the deadlocked memory port will block until manually cleared. This situation was often caused by poorly designed I/O controllers that recovered from errors by simply aborting the current access with no regard for proper termination of UNIBUS or switch protocols.

While the known cases that deadlocked memory ports were isolated and individually remedied, the most important result was an appreciation of the design principle of mutual suspicion [8]. The switch should never trust that an operation started will necessarily be completed; it must be prepared to time-out, clear itself, and report a failure condition to the requesting processor.

The interprocessor bus is as unreliable as the switch is trustworthy. Its reliability is so poor that if a cheap and highly effective method of software recovery (discussed in the next section) hadn't been found, the bus would be nearly unusable. The mode of failure is transient loss of interprocessor interrupts and changing interrupt levels - usually from level 7 to level 6. No cause has been isolated.

Two remaining long-term reliability artifacts of the architecture are:

1. Overrun errors on I/O device DMA transfers caused by memory port contention. This is a predictable result of not having the planned cache memories and is effectively recovered from in software.

2. Having I/O devices associated with specific processors causes undesirable dependency on that processor. A partial solution has been developed in software to recover from transient failures, but frequent or hard failures force a shutdown for repair or reconfiguration. Fortunately, shutdown is very rare.

2.2.3 Software Recovery Methods Within the Hydra Kernel

As the above description of the failures encountered indicates, fault tolerance is the result of a highly cooperative effort between hardware and software. Some failures, such as losing interprocessor interrupts, produce no damage and require so little effort in software recovery that little motivation exists to correct the hardware. Others (deadlocked memory ports) are impossible to recover from with software and much manpower has been devoted to eliminating the sources of failure. The software recovery methods, developed by design and evolution, may be similarly grouped: methods for recovery from frequent failures that have little probability of non-local damage, and methods for treating relatively rare, but serious, failures that may imply system-wide damage.

Examples of the first class of failures are typically transient, though frequent, and do not involve shared data structures. The recovery methods were developed to suit each case; three such cases will be examined in detail: interprocessor interrupt failures, DMA overruns, and memory parity failures in user-allocated pages.

Interprocessor interrupt requests are recorded in a software mask. When an interrupt is to be sent, the bit(s) corresponding to the processor(s) to be interrupted are set in the mask for the interrupt type. The mask rather than just the request bit(s) is then copied into the interrupt request register. Upon receipt of an interrupt each processor checks the mask for its bit. If the bit is set, it is then cleared and the interrupt processed; if the bit is not set, the interrupt is considered redundant and ignored. A lost interrupt is then repeated automatically by the next request since the bit has not been cleared from the mask. The

frequency of interrupt requests assures that a lost interrupt will be repeated quickly, likely by a different processor. Priority-shifted interrupts become redundant interrupts at the arrival level and are ignored. The next request for an interrupt at the intended level will automatically repeat the original request. The method is cheap, requiring only four instructions: a bit-set on transmission and a bit-test, branch, bit-clear sequence on receipt.

DMA overruns present a problem when the system is heavily loaded. The device handlers attempt five retries as part of the standard device failure recovery before invoking overrun recovery. Recovering from repeated overruns requires gaining exclusive access to the required memory port. Scheduling interrupts are sent to all processors except the requestor with instructions to execute a timed loop in local memory (Mlocal). Execution of this loop eliminates all shared memory accesses giving the device the access it needs. The timing of the loop allows the largest DMA transfer to complete. Upon completion of the loop, all processors resume normal execution. Since the system is required to pause for the transfer, the method is not cheap, but the condition is sufficiently infrequent to make the cost acceptable. Completion of the cache memories will alleviate the overrun condition, but there may still exist rare circumstances that may require that this recovery scheme be retained.

Memory parity failures in user-allocated pages are reported to the user process via its error trap routine, as are NXM and illegal instruction exceptions. As the tracking registers are added to the relocation units, detailed information about the failing location and location of the instruction being executed will be passed to the routine. These registers will also allow the paging system to restore a logical page to another physical page frame if a valid copy exists on secondary memory. Reporting errors to the user process is an example of a design decision: we always attempt to reflect exceptions back to a level where there is sufficient information for proper action [9].

Since Hydra is only the kernel of the operating system [4], important system elements such as job scheduling and file systems are implemented as user-level programs. Their response to error traps such as above is highly variable and beyond the scope of this article, but many have a common technique, using multiple processes. These processes may be multiple incarnations of the subsystem's server processes, or they may be free-running "daemon" processes created specifically to play a watchdog role in insuring the correct and reliable operation of the subsystem. The multiple incarnations approach accepts the loss of a server and the processes dependent upon it as a method of limiting damage. It also tends to improve response. The daemon approach is specifically creating redundancy for reliability.

For the second class of errors, those that imply critical, system-wide damage, a formal mechanism is invoked. This mechanism, the suspect/monitor model, causes the whole system to pause so that a known state is reached before a sequence of error logging and analysis is performed. This procedure allows a wide range of options, from continuing execution, possibly with configuration changes, to reloading (possibly reconfigured). Developed in response to the low reliability of the developing hardware and software, suspect/monitor was retrofitted to the existing software. Since most data structures lack the redundancy and associated verification routines to guarantee repair of damage, all paths through suspect/monitor currently lead to system reload.

A description of the suspect/monitor sequence follows. Invocation occurs in two ways. First, a processor may detect an error condition either by hardware trap or software check. It then becomes the suspect and a monitoring processor is randomly chosen from the remaining processors. Second, a processor executing the "watchdog" routine detects that some other processor has apparently not been executing. The watchdog processor becomes the monitor and declares the apparently non-executing processor to be the suspect. The

watchdog routine is executed by all processors as part of several frequently used interrupt service routines and sets a bit (corresponding to the executing processor) in a mask maintained by the watchdog. Periodically this mask is compared with a mask of processors known to have completed initialization (upmask) and then cleared. Any processors in the upmask but not in the watchdog mask are declared suspects.

Once the monitor is chosen, it and the suspect must achieve synchronization. A shared state variable is used to communicate between suspect and monitor. Each advance the variable to the next state upon entry. Both examine the state and if it is not in the synchronized state, they wait for the other to advance it to that state. The monitor times all waits for the suspect to reach a desired state. In this case, if synchronization is not achieved quickly, the monitor attempts to force the suspect processor to execute the recovery code with a sequence of interprocessor bus operations. Continued failure to synchronize causes the monitor to abort the sequence and force a reload. Multiple suspects are processed one at a time by the same monitor.

The suspect processor's sequence is: record all processor state at the time of failure, including which pages were addressable, copy its local memory, execute a short diagnostic, and assuming correct execution of the diagnostic, attempt analysis of the failure. Completion of these actions is communicated to the monitor via the shared state variable. Because of the sensitive nature of the suspect's execution, several coding restrictions were employed in its implementation. No stack operations are performed since they are failure prone, no loops are allowed so the processor state logging code is straight-line and fast, a flag is set upon entry to the suspect routine to force an immediate halt upon re-entry for any reason. Halting causes the monitor to force a reload and prevents the previously logged data from being overwritten.

Once synchronized, the monitor follows the suspect through its sequence. If the suspect fails to complete any operation in the allotted time, the monitor forces a reload. After a suspect completes its sequence, the monitor has the following options:

- continue with no changes
- halt the suspect and continue
- "quiesce" the suspect and continue
- reload
- reload, delete suspect from configuration
- reload, "quiesce" the suspect

"Quiescing" a processor allows it to service I/O device interrupts, but not execute any other functions (notably user programs). By only allowing I/O interrupt processing, the duty cycle is kept low and, hopefully, so is the probability of a failure. This mode is required to keep processors with critical I/O devices in the configuration.

The analysis that the suspect may perform is highly failure dependent. Due to the problems of installing any recovery scheme in an existing large program, the problems of analysis are only beginning to be examined. Recovery from memory parity failures during kernel execution is being considered as the first candidate for analytical recovery. These parity failures are considered serious enough to invoke suspect/monitor because the abstract

data type system, the heart of Hydra's protection system, must be as reliable as possible [4], [6]. Further, a page may hold segments of many abstract data objects so a failure may imply future damage if not caught promptly. For parity failures, the analysis must determine three facts: if the failure is repeatable, if it happened during interrupt service, and if any critical data structures were locked. If any of these are true, recovery is not possible. There is no way to report the failure to the process while servicing an interrupt. If locked, a data structure may be in an inconsistent state. In these cases, the suspect notifies the monitor to reload the system. Otherwise, the failure has occured during a kernel call and may be aborted with a parity failure report. It is then the responsibility of the caller to decide whether to retry the call. No claim is made that this particular method is optimal; it is intended to illustrate the role of analysis in the suspect/monitor. However, it does promise a high probability of recovering from the majority of parity failures with an acceptably small risk of undetected damage.

The auto-restart mechanism is responsible for reloading the system. Three basic steps are involved: adjusting the configuration masks for any deleted or quiesced processors, constructing a free memory list (deleting pages that have been marked errant), and loading a fresh copy of the kernel from disk. The new system is entered and initialization begins. This sequence is normally accomplished without human intervention.

The last mechanism associated with failure recovery is the automatic diagnostic driver which initiates and monitors the deleted processors' execution of a diagnostic. The driver maintains a history of the failures found by each processor as well as their successful executions of the diagnostic. The histories may be printed on command and are also accessable from Hydra. If a processor is able to successfully run the diagnostic for a period of time determined by its failure history over the previous few days, the driver automatically returns it to the system. Automatic return is accomplished by executing the per-processor initialization and does not require pausing or reloading the system.

2.2.3.1 Conclusion

The success of the error detection and recovery methods in Hydra is considered one of the project's more notable accomplishments [7]. Fault tolerant methodologies will continue to be a prime research area in the future; the current success is considered just a beginning.

A short antecdote will conclude this section in a light vein and illustrate the effectiveness of the error coverage and restart mechanisms. Late one Friday night, a power failure shut down all the machines. Several of the larger machines suffered damage. C.mmp was not spared: a large power supply in the switch was lost causing half of the memory to become inaccessable. This massive fault overwhelmed the auto-restart system (undoubtably one of its pages was lost) and the system lay quiet after power was restored until one of the users grew tired of waiting. He followed the simple tape restart instructions posted on the machine (C.mmp does not have, or need, an operator) and the system scanned memory, found what was left, initialized itself, and announced that it was ready for use. The user went back to work without an inkling of the machine's loss. In fact, none of the Hydrants knew until the following Monday morning when the system engineer came in and said "Say! Do you realize...?"

2.3 C.mmp Reliability Data

The data presented here were culled from the crash reports produced by the Hydra suspect/monitor crash logging system. These dumps must often be manually analysed to determine the reason for the crash. Sometimes the reason cannot be found; always the

analysis is error-prone, being a manual procedure. The crash records were never intended as a precision reliability measure, rather, they are a programmer's and engineer's tool to isolate trouble spots in the system. With this caveat in mind, the data may be discussed.

2.3.1 Interpreting the Data

A failure causing a crash may be the result of either hardware or software malfunction. Of the five symptoms listed in Table 1, only parity failures are necessarilly caused by hardware. All the others may be brought about by either and analysis is required to determine the actual cause. The cause of most failures can be determined, but a substantial number of crashes of unknown origin remain.

Examining Table 1, a trend is apparent. Parity failures are the major source (50% to 100%) of all hardware-related failures. The other significant sources are NXM and non-response. Analysis of many crashes has shown both of these to often be memory-related.

Errors due to software follow entirely different patterns. The error frequency is strongly related to the introduction of new features. Being new and relatively untested, new features are likely to have previously undetected faults. Once the feature is installed, any errors due to it are usually found and corrected very quickly. Therefore, the trend is bursts of errors, any particular error becoming less frequent as time passes. The four months with high software error counts all follow this trend even though new faults kept the counts high for several months running.

The lack of independence among the symptoms, while present in all complex systems, is increased by the lack of fault tracing facilities in C.mmp. All reliability measurements tend to measure large sections of the system. Consequently they are coarse-grained and uncertain.

2.3.2 Eight Months of Data

Month	July(1)	Aug. (2)	Sep. (3)	Oct. (4)	Nov. (5)	Dec. (6)	Jan. (7)	Feb. (8)
Year	1977	1977	1977	1977	1977	1977	1978	1978
Up time	516.6	610.5	513.8	701.9	538.8	595.6	600.2	478.5
MTBF	5.9	7.6	2.9	9.4	8.7	16.5	15.4	7.3
Crashes								
User	32	55	38	27	34	18	15	30
NonUser	87	80	175	75	62	36	39	66
Crash Type								
SoftWr	20	7	35	33	34	11	7	16
Unknown	32	40	14	4	9	7	8	3
HardWr	35	33	126	38	18	18	24	47
Crash Symptom								
Syserr	24	10	47	46	31	11	9	15
IllInst	1	0	3	3	0	2	0	0
NoResp	13	33	34	3	4	4	10	10
NXM	14	13	32	4	9	5	2	14
Parity	32	24	57	17	18	14	18	21

MTBF = (Up time)/(NonUser crashes)
IllInst = Illegal Instruction

Table 1 A Summary of 8 Months of C.mmp Crash Data

Some comments about the data:

1. The software error totals for July, September, October, and November are, with one execption, due to different causes each month.

   - July: local memory overwritten (13 out of 20 (13/20) crashes),

   - Sept.: microcode verifier bug (27/35 crashes),

   - Oct.: critical section count bug (14/33) and drum directory full (15/33)

   - Nov.: Hydra message port create bug (9/34), paging system bug (11/34), and drum directory full (11/34)

2. The SYSERR (software detected errors) count is nearly always greater than the number of software-caused crashes. These are examples of the software detecting hardware failures not caught by the hardware itself.

The high counts for errors of unknown origin in July and August are due to training a new person in the arcane arts of crash dump analysis. After his trial by fire in September, he became much better, but this only underscores the basic uncertianty in manual analysis.

Figure 3 graphically restates the data from Table 1 to show the contribution of each of five classes of errors. The "glitch" at point 5 (November) is due to some of the NXM's and non-response errors being due to software. This again illustrates the impossibility of defining independent error classes on C.mmp.

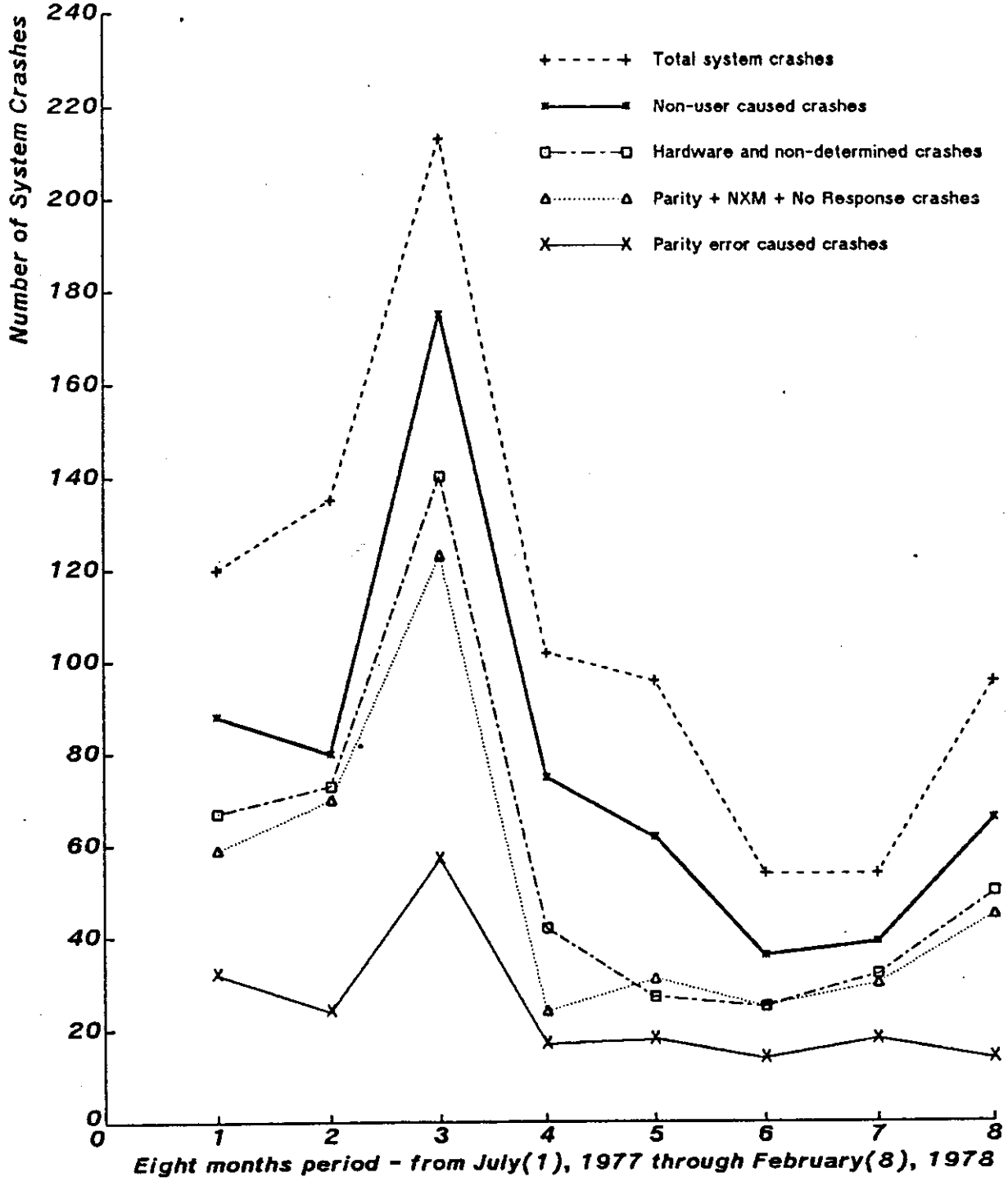


Figure 3 C.mmp Reliability - distribution of crashes

3. Cm*: A Modular Multi Microprocessor

Cm* was designed to take advantage of increasing complexity of LSI technology The original goals included modularity, effective performance/cost ratio and reliability [10]. The Cm* architecture allows close cooperation between large numbers of inexpensive processors. All processors share access to a single virtual memory address space. A ten processor system has been operational since May 1977 [11] and expansion to a 50 processor system should be completed by the end of 1978.

Some of the features of this architecture include:

Extensibility. There are no fundamental limits of the size of the system. Processors, memories and interconnects can be incrementally added to increase processing power, memory size and communication bandwidth. The system topology can be constructed to match individual applications. [12] [13]

Address Mapping. All memory in the system is potentially accessible to all processes. A sophisticated mapping from processor generated addresses to physical memory addresses provides the means for memory sharing. An extensible set of interprocess control mechanisms is constructed within the address mapping structure. [14]

Operating System Primitives. An interprocessor message system is supported by the Cm* hardware. The writable control store allows experimentation and extension of firmware primitives. [15] [16]

Cost/Effectiveness. The interconnection structure allows large numbers of (potentially mass produced) inexpensive digital modules to share resources and cooperate on large computation tasks. [17] [18]

Reliability. Distributed intelligence in the form of processors and communication interconnects means that there is no critical system resource whose loss would cause system failure. Changing of the address mapping functions allow pruning of faulty components. Parity, remote diagnosis, and instruction retry allow the detection and correction of transient and permanent faults. [1] [19]

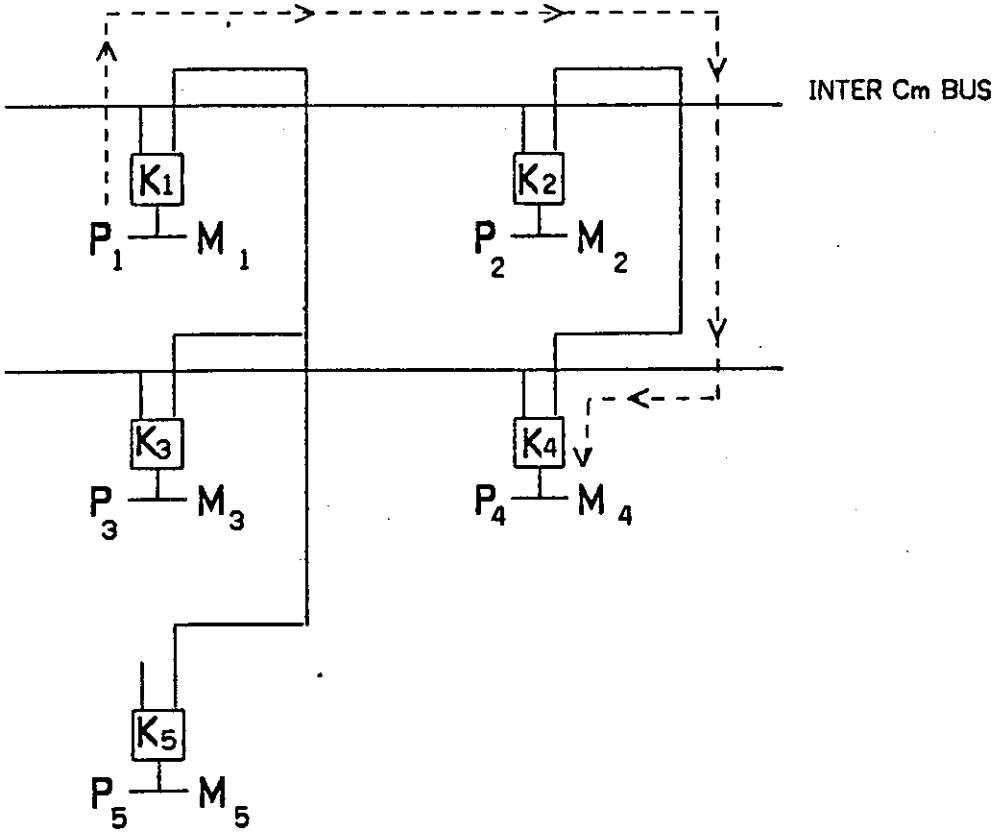


Figure 4a. Canonical Computer Module Structure

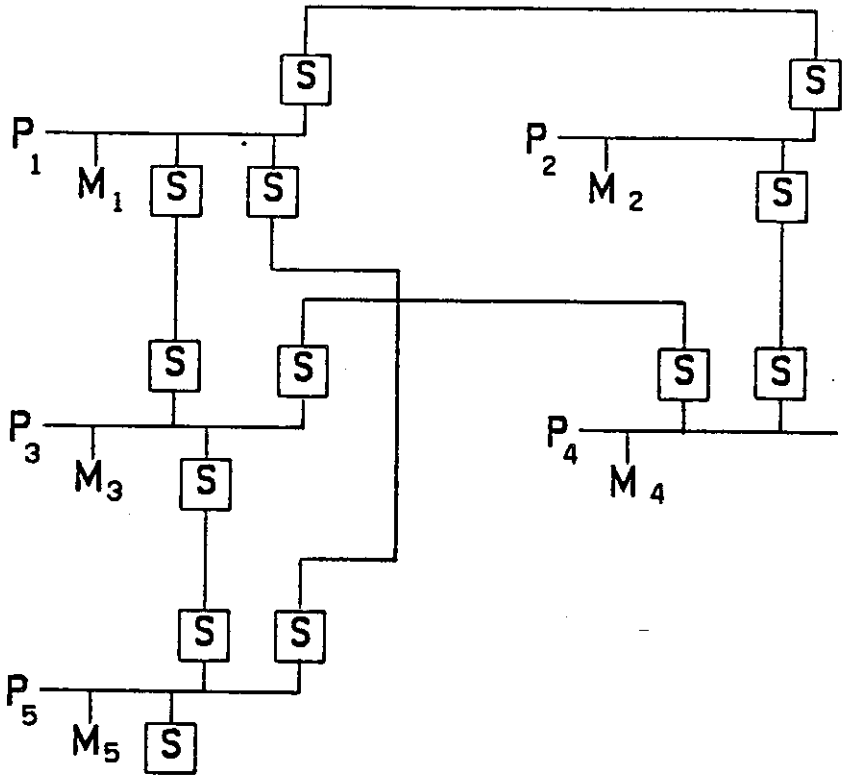


Figure 4b. Module Interconnection with Simple Links

Design	Chips per Port	External Connections	Performance degradation factor for a nonlocal memory read operation with 1 level of mapping	Representative Fully Connected System		
				10 Cms	100 Cms	Total Chips (10 Cms)
K.map/Cm (Fig. 4a)	380	25 to 75	1.5	10 K.maps	100 K.maps	3800
Simple Links (Fig. 4b)	25	24 to 240	8	180 Links	19,800 Links	4500
K.map/Cluster (Fig. 7)	125 (S.local) 650(centralized K.map)	26 (26 to 78 per K.map)	1.8	10 S.locals 1 K.map	100 S.locals 10 K.maps	1900

Figure 5. Comparison of Three Alternative Implementations

3.1. The PMS Structure of Cm*

The structure of Cm* has developed over a number of years into the "canonical" structure in Figure 4a [10] [20] [21] [22]. This is a structure with a low concurrency switch (the network of buses) giving access to shared memory. The structure is built from Processor-Memory pairs called Computer Modules or Cm's. The memory local to a processor is also the shared memory in the system. Inherent in this structure is the assumption of program locality. The efficient use of the system depends on ensuring that most of the code and data referenced by a processor will be held local to that processor. Early measurements with various benchmark applications indicate that local memory hit ratios of 0.8 to 0.9 (i.e., ratio of time memory reference is to local rather than remote shared memory) are readily achievable [11].

3.2. Derivation of Cm* Structure [11]

A series of design studies were undertaken to explore this design space. Figure 4 depicts the various PMS structures studied, while Figure 5 lists the estimated cost and complexity of each design [22]. Figure 5 was created during a preliminary design exercise, hence the numbers are only approximate figures.

Initially, we envisaged one self-contained module which consisted of a processor, memory and an intelligent interface (Figure 4a). The result was termed a computer module (Cm). The mapping controller (K.map, marked K in the figures) performed all the functions necessary for generating external memory requests and responding to external requests for its local memory. So that the capacity for interprocessor communication would not be limited by any single communication path, each K.map connects to three inter-Cm buses. Memory could be shared even though there was no direct physical connection between the requesting processor and requested memory. For example, consider a request by P1 to M4 in Figure 4a. K.map1 would route the request to K.map2 which would route it, in turn, to K.map4. From Figure 5 we see that this design was extremely costly while a simulation/benchmark study indicated that the bus structure was under-utilized. Subsequently we tried as simple a design as possible in order to minimize the complexity. Figure 4b depicts the simple interface (S.minimal, marked S in the figure) design. The minimal interface provided parallel word transfer between two buses. Every pair of Cm's that required direct physical links could communicate via intermediate modules provided the delays for the intermediate passing of requests were acceptable. From Figure 5 we see that the projected performance was low and that for fully interconnected structures the cost was comparable to the K.map per Cm scheme (Figure 4a).

Our investigation led us to the conclusion that very little performance loss resulted from centralizing the address mapping and multiple bus connection functions of individual modules in a K.map which is shared by a number of computer modules. The cost savings are quite dramatic. Figure 5 shows a saving of a factor of two in chip count for comparable structures. The cost savings are better than indicated by Figure

5, because the final shared K.map design incorporates many features not accounted for in the chip counts for the other designs, for example 30K bytes of bipolar RAM for microcode and data storage. The programmable, high performance K.map is shared by several Cm's connected to an inter-Cm bus via simple interfaces (S.local). The basic function of the S.local is to provide a buffer between the processor and the inter-Cm bus and sufficient control functions to generate or respond to external memory requests.

Figure 6 shows the details of a Computer Module. The processor is a DEC LSI-11, which is program compatible with the PDP-11 family but is implemented with LSI technology and uses a cheaper and lower performance memory bus. Figure 7 shows a cluster of Computer Modules sharing a single map bus and mapping processor, or K.map. Figure 8 shows the third hierarchy in a Cm* structure: Cm clusters connected via intercluster buses.

The advantages of sharing a mapping processor across a cluster of Cm's are much broader than the simple chip count advantage indicated by Figure 5. Because the cost of the K.map is distributed across many processors it can be endowed with considerable flexibility and power at relatively little incremental cost. Because of its commanding position in the cluster, the K.map can ensure mutual exclusion on access to shared data structures with very little overhead. Further, the K.map can monitor Cm and intercluster activity during normal operation. From this a constantly updating picture of process activity and malfunctions are created for use in load balancing and system recover/reconfiguration.

The K.map, in addition to arbitrating and controlling the map bus, is a horizontally microcoded 150ns cycle time processor. The basic configuration has 2Kx80 bits of writable control store and 5Kx16 bits of bipolar RAM for holding mapping tables, etc. A Linc provides the interface to two intercluster buses. The K.map has many features which tailor it to the task of address mapping.

In addition to address mapping and the routing of requests to other clusters, the K.map provides a powerful protection mechanism and a low overhead interprocess communication message system.

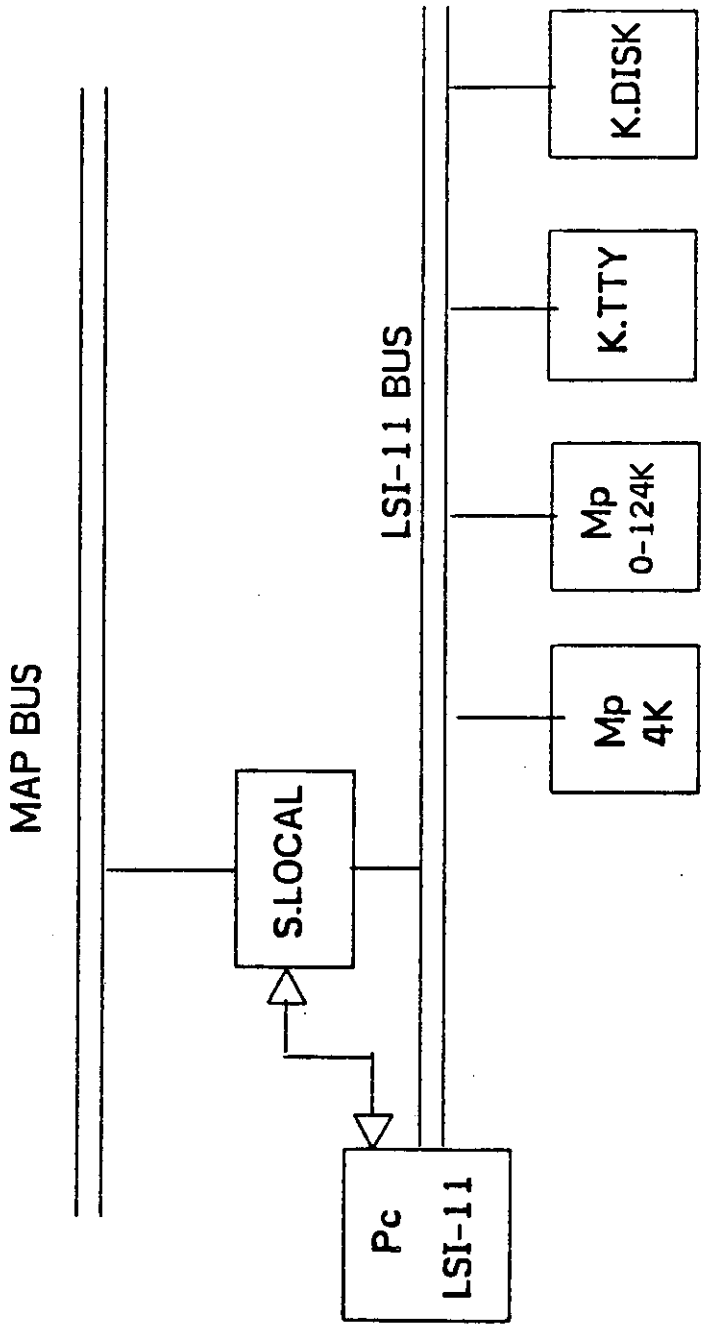


Figure 6. Details of a Computer Module

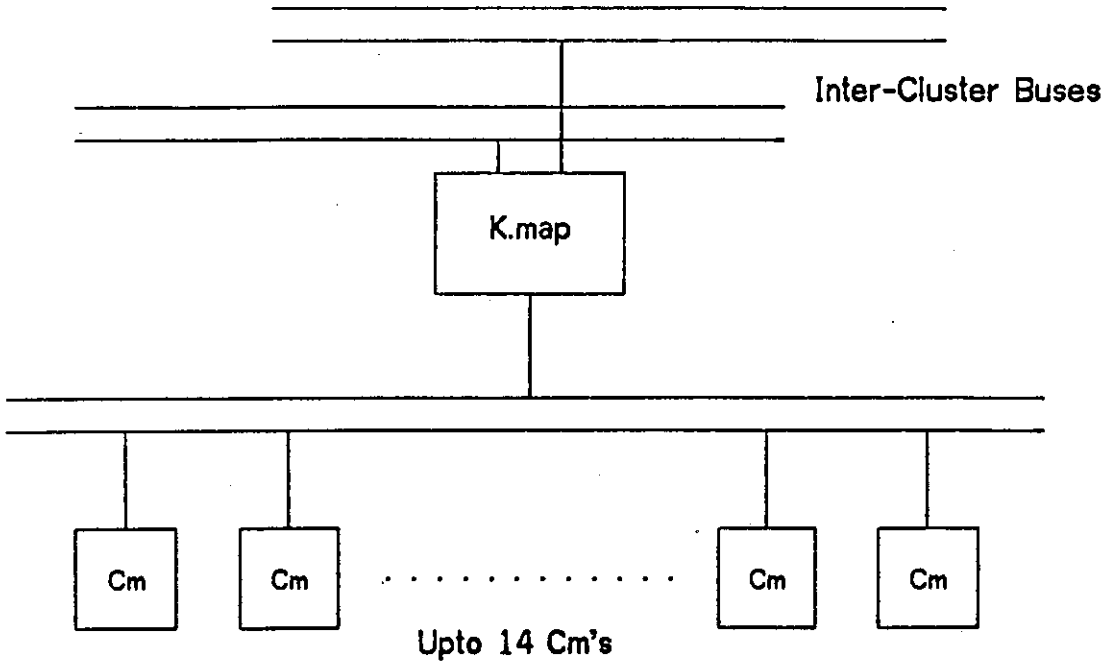


Figure 7. A Cm* Cluster

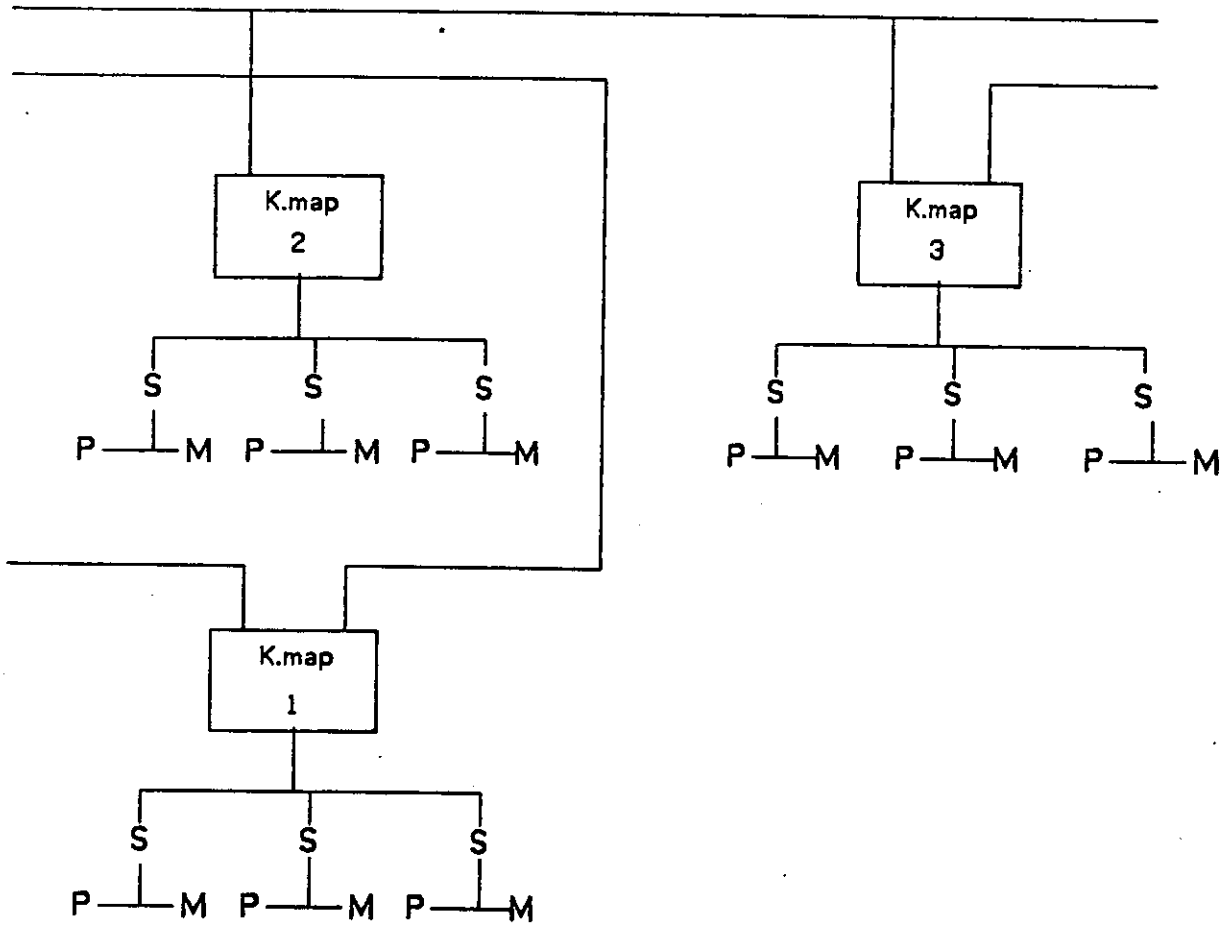


Figure 8. Simple Three Cluster Network

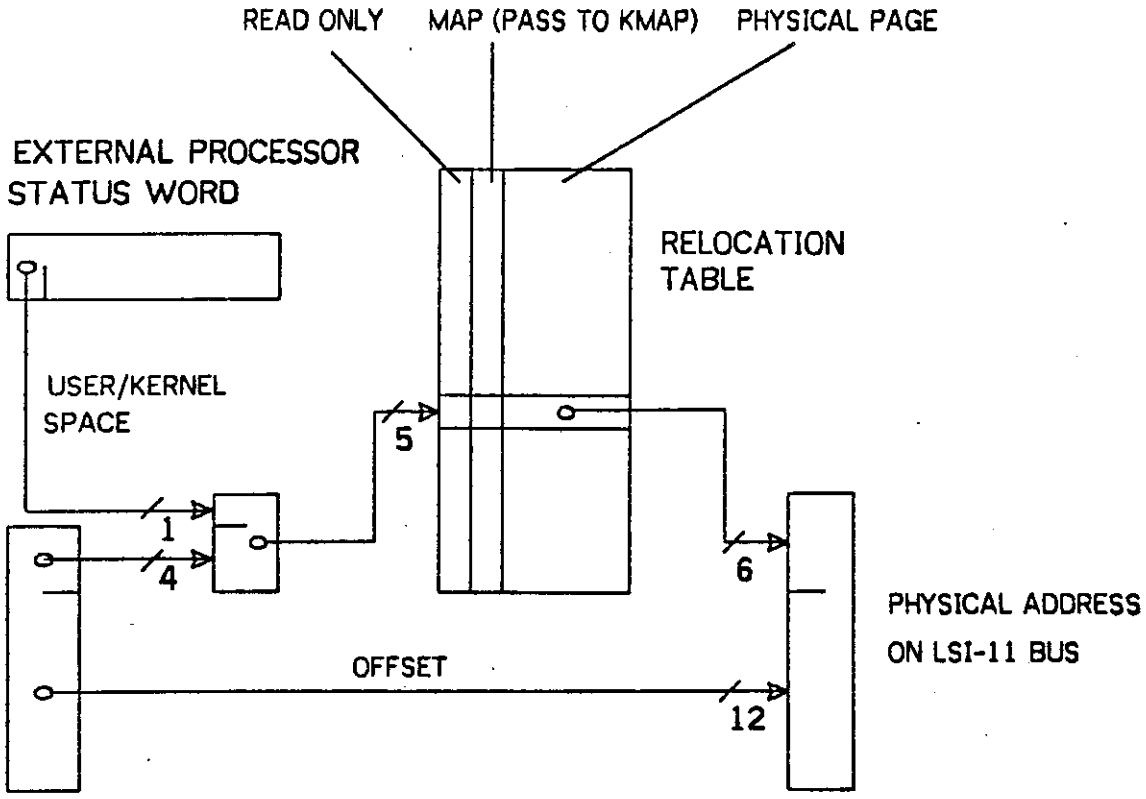


Figure 9. Addressing Mechanism for Local Memory References

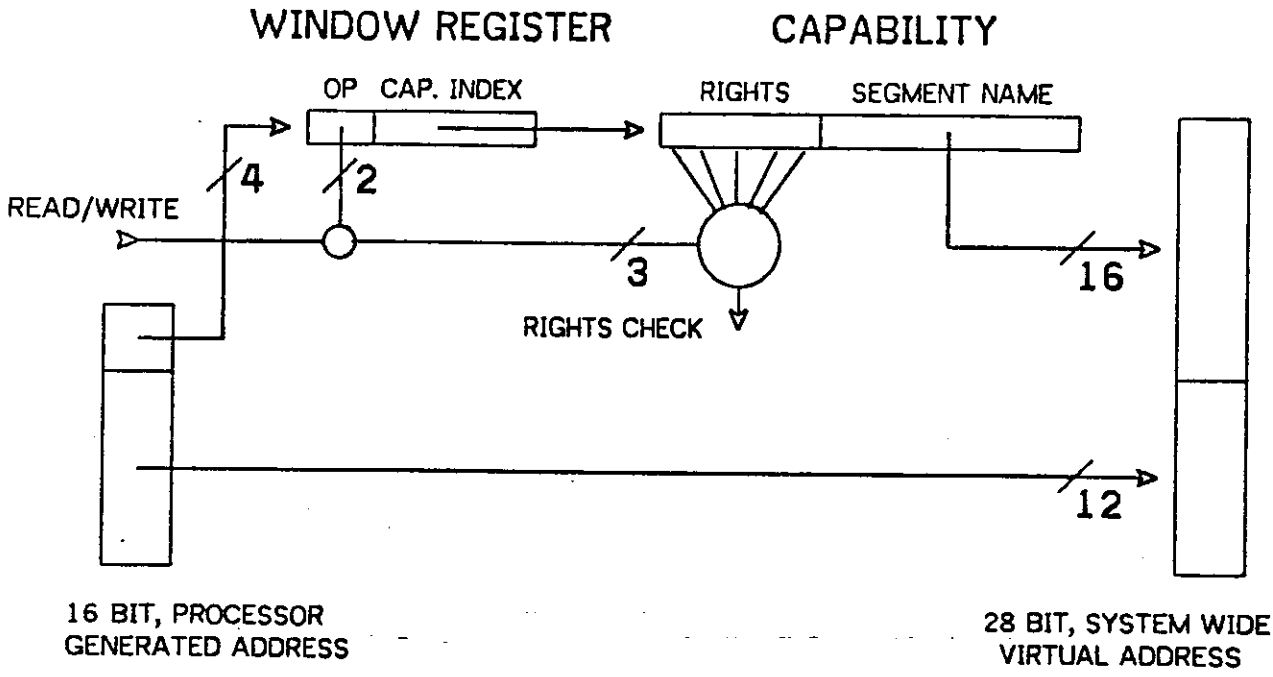


Figure 10. Conceptual Virtual Address Generation and Rights Checking

3.3. Address Mapping in Cm*

Cm* has a 2^{28} byte segmented virtual address space. Segments are of variable size up to a maximum of 4K bytes. There is a capability-based protection scheme enforced by the Kmap. The addressing structure provides considerable support for operating system primitives such as context switching and interprocess message transmission.

3.3.1 The Path from Processor to Memory

The Slocal (see Figure 9) provides the first level of memory mapping. A reference to local memory is simply relocated, on 4K byte page boundaries, by the relocation table in the Slocal. As discussed above, it is assumed that most memory references will be made by the processors to their local memory. Relocation of local memory references can be implemented with no performance overhead because the synchronous processor has sufficiently wide timing margins at the points where address relocation is performed. For segments which are not in a processor's local memory the relocation table has a status bit which causes the address to be latched, the processor forced off the LSI-11 bus, and a Service Request to be signalled to the Kmap. All transactions on the Map bus are controlled by the Map bus controller, or Kbus, which is a component of the Kmap. The address generated by the processor is transferred bia the Map bus

to the Pmap, the microprogrammed processor within the Kmap. If the reference is for memory within the cluster then the Pmap generates a physical address and sends it to the appropriate Slocal. If it is a write operation, data is passed directly from the source Slocal to the destination Slocal; the data does not have to be routed through the Kmap. The selected destination Slocal performs the requested memory reference and the processor in the destination Computer module is not involved. When the reference is complete the Kbus transfers the data read from the destination Slocal directly back to the requesting processor via the Map bus and its Slocal.

If the processor references a segment in another cluster then the Pmap will transmit a request to the desired cluster via the Linc and the Intercluster buses. If the destination cluster does not share a common Intercluster bus with the source then the message will be automatically routed via intermediate clusters. When the message reaches the destination cluster, the memory reference is performed similar to a request from a processor within the cluster. An acknowledgement, or Return, message (containing data in case of a read) is always sent back to the source cluster and subsequently to the requesting processor.

Several points are worth noting here with respect to the mechanisms for local and non-local references. Except at the local memory bus level, where conventional circuit switching is used, all communication is performed by packet switching. That is, busses are allocated only for the period required to transfer data. The data is latched at each interface, rather than establishing a continuous circuit from the source to the destination. This approach gives greater bus utilization and avoids deadlock over bus allocation. All transactions are completely interlocked with positive acknowledgement being required to signal completion of an operation (it is possible to allow a processor executing a nonlocal write to proceed as soon as the data for the write has been received by the Kmap or the destination Slocal, without waiting for completion of the operation; however, in this case the Kmap will expect to receive acknowledgement in place of the processor so that appropriate actions may be taken if none is received).

3.3.2 The Addressing Environment of a Process

The virtual address space of Cm* is subdivided into up to 2^{16} Segments. Each segment is defined by a Segment Descriptor. The standard type of segment is similar to segments in other computer systems; it is simply a vector of memory locations. The segment descriptor specifies the physical base address of the segment and the length of the segment. Segments are variable in size from 2 bytes to 4K bytes. However, other segment types may be more than simple linear vectors of memory; references to segments may invoke special operations. Segments may have the properties of stacks, queues or other data structures. Some segments may not have any memory associated with them, and a reference to the segment would invoke a control operation. For each segment type, up to eight distinct operations can be defined. For normal segments the operations are Read and Write. Conceptually, segments are never addressed directly; they are always referenced indirectly via a Capability. A capability is a two-word item containing the name of a segment and a Rights field. Each bit in the rights field indicates whether the corresponding operation is permitted on the segment.

3.3.3 Virtual Address Generation

The processors in Cm*, LSI-11s, can directly generate only a 16 bit address. This 64K byte address space is divided into 16 pages of 4K bytes each. Each page provides a window into the system-wide 2^{28} byte virtual address space and can be independently bound to different segments in the virtual address space. The top page in the processor's address space, page 15, is reserved for direct program interaction with the Kmap. The mechanism is analogous to the I/O page convention in the standard PDP-11s.

Figure 10 shows the conceptual translation from a 16 bit processor-generated address to a virtual address. The four high order address bits from the processor select one of 15 Window registers (which are pseudo-registers in page 15 and bind page frames in processor immediate address space to segments in virtual address space). The Window register holds an index for a Capability in the executing software module's Capability List structure. The 16 bit segment name from the selected Capability is concatenated with the 12 low order bits from the processor to form a 28 bit virtual address. Figure 10 also shows the read/write indicator from the processor being concatenated with the two bits in the address expansion registers to form a three bit opcode. The corresponding bit in the Capability rights field is selected and tested. If the operation is not permitted then an error trap is forced.

3.3.4 The Kernel Address Space

Each processor can execute in either of two address spaces. One is the User Address Space which was described above. The second is the Kernel Address Space, which is quite similar to a user address space with the addition of some mechanisms reserved for the operating system. The currently executing address space is selected by a bit in the Processor Status Word of the LSI-11.

3.4. Development and Diagnostic Aids in Cm* Hardware

Simulators for hardware are expensive both in terms of development effort and computer time; furthermore they cannot give an exact reflection of the hardware. Thus this approach leaves the final bugs to be found using the real hardware, and is of no aid in diagnosing component failures (rather than design errors). The alternative approach adopted for Cm* was to incorporate special hardware, called Hooks, directly into the Kmap for use in hardware and microcode development [13]. The interfacing of the Hooks to a standard LSI-11 allows extensive software support for hardware development and diagnostics while at the same time providing a convenient environment for the debugging of microcode on the real hardware.

The Hooks give to an LSI-11, referred to as the Hooks Processor, the ability to intimately examine and change the internal state of the Kmap. They provide the capability for the Hooks Processor to load the microcode into the writable control store of the Pmap, read the values on the busses of the Pmap, and to independently

start, stop, and single-cycle the Pmap-Linc and Kbus clocks. An interrupt is generated for the Hooks Processor whenever the Pmap clock stops (either due to a microprogram-invoked halt or a memory parity error on the control or data stores). Furthermore, several of the internal registers of the Pmap have "twin registers" associated with them which may only be loaded by the Hooks Processor. These alternate registers may be enabled via the Hooks to override microprogram-controlled values. The presence of the Hooks added approximately ten percent to the cost of the Pmap while enormously reducing system development time.

Several registers in the Slocal are concerned with providing diagnosis and recovery information after a software or hardware error is detected. Almost all errors are reported to the processor by forcing a NXM (Non eXistent Memory) trap. This includes errors detected by the Kmap during remote references. The Kmap signals the error by writing to the "Force NXM" bit in an addressable register in the Slocal. The Local Error Register indicates the nature of the error and whether the erroneous reference was mapped. The "Last Fetch Address" register is updated to hold the address of the first word of an instruction every time the LSI-11 fetches a new instruction. If an error is detected, this register is frozen until the Local Error Register is explicitly cleared. Also frozen in the Local Error Register is a count of the number of memory references performed in the execution of the instruction. In conjunction, these two registers provide sufficient information to restore the state of the LSI-11 for retry of the instruction during which the error was detected.

All external memory accesses are performed by message switching. Each message is buffered until receipt of a positive acknowledgement. Upon time-out or an error message, the Kmap attempts to retransmit the message, possibly over a physically different path. The user is only notified if the destination memory has been isolated or has a hard failure.

All system memory is protected by parity. When a parity error occurs the address is captured and the Kmap notified so that it might retry the memory access. Table 2 lists the parity and time-out features.

Data path parity and message switching combined allow single bit error detection as well as correction. The message is passed along with vertical parity. The receiver checks both horizontal and vertical parity; if there is a single error the intersection of the two parity bits will uniquely indentify the erroneous bit.

Most errors in the system will apear as memory access errors (e.g. parity, time-out, or capability violation). Thus the operating system can focus on recovery from a single class of errors rather than treating numerous special cases.

Parity

Subsystem	Where Error Reported
Local Memory	Error Trap to Requesting Processor and Interrupt to Local Processor
Map Bus	To buffered context in K.map
K.map	Hooks Processor
Data RAM	
Code RAM	
L.inc Memory RAM	Local K.map
L.inc Intercluster Bus	Local K.map

Timeouts

Subsystem	Duration	Where Reported
Local Memory	12 usec.	Local Processor
S.local (source)	200 msec.	Local Processor
S.local (destination)	12 usec.	Report Non-existent memory back
K.map	2-4 msec.	Report error message back to source
L.inc	20 usec./word	Local K.map

Table 2, Parity and Timeouts in Cm*

3.5. Autodiagnostic Software for Cm* [23]

Autodiagnostic software was developed to exercize idle modules in Cm*. It is designed to run on any Cm which has a serial line unit to the Cm* Host, a message switching PDP-11 [24]. The Autodiagnostics process runs continuously and operates as follows.

The initial step taken by the Autodiagnostics is to signal the Host that it wishes to be considered a master. Once in the master state it appears to the Host as any other terminal. This fact enables the Autodiagnostics to log into the system as a user and request a system status report regarding the serial lines into the host. The report indicates which modules are operational in the system and whether they are slaves or masters. Once the configuration is known, the Autodiagnostics requests the Host to assign to it any Cms which may be idle. It then proceeds to load these Cms with diagnostic programs and start them. The Autodiagnostics requests assignations of idle Cms every seven minutes. If a user should request resources, however, it relinquishes Cms that have been assigned to it.

There were three constraints on the design of the Autodiagnostics. First, off-the-shelf DEC diagnostics were to be used. These diagnostics were used unmodified since no source code was available. Second, the diagnostics written for the CMU built hardware were subject to change. Third, initial tests indicated that if enough modules

were executing diagnostics which constantly printed diagnostic messages, the Host-Autodiagnostic connection could be saturated causing loss of characters.

Diagnostics are programs that are designed to detect hard faults, however, some transient failures can also be detected by them. Diagnostics for exercizing various parts of the system are loaded sequentially. The condition for loading the next diagnostic program into an idle Cm is either the successful completion of a specified number of passes of the diagnostic or the occurrence of a specified number of failures. In the latter instance attempts are made to obtain as much information as possible from all the diagnostics about the failure condition.

The Autodiagnostics must be able to automatically handle certain situations. One such problem is detecting that a module is stuck in an endless loop. If a module has not responded in a given time, it is reloaded and started. Another problem is that of timeouts; these are detected if a character that is expected from the Host is not received in a given time. A timeout message is printed and the task being performed is put on the queue for a retry.

There are two types of information that the Autodiagnostics delivers. The first consists of error messages that are output whenever a module detects an error. These messages include the identity of the module, the name of the diagnostic being run, the current time so the error can be time-stamped for future reference, the time since the last error on the module, and the error information extracted from the diagnostic.

The second type of information consists of status reports which include the time the Autodiagnostics were started, the time of the report, and status information on all the modules. This information is output in two tables. The first is a general report that gives the overall length of time the diagnostics were run and the number of errors detected for each module. The other table breaks this information down further by the diagnostics. A summary of the total module hours and module errors for the system are also included.

3.5.1 Diagnostics

A set of four diagnostics are continuously run on the Cms. These tests exercise (i) the memory, (ii) instruction set, (iii) traps and interrupts, and (iv) the Slocal and a small part of the Kmap.

The memory test is divided into 13 subtests, which include a gallop test, marching ones and zeros, and shifting [25]. It takes approximately 13 minutes to complete one pass through 56K bytes of dynamic MOS RAM.

The instruction set [26] and the traps and interrupt diagnostics [27] are designed to test the functioning of the LSI-11 processor. These are short tests so many passes are done before moving on to the next diagnostic.

The Slocal diagnostic preforms a number of functions. First it tests the registers and data paths of the Slocal. Second, it exercises a part of the Kmap. Finally it runs a few tests on portions of memory.

3.6. Data on Transient Errors

In the Cm* system, the Autodiagnostics uses diagnostic programs to detect transient errors. It is known that transient errors are more likely to be detected by certain sequences of tests than others. Since the diagnostics used were designed to detect hard failures in the system it is realized that they are incapable of detecting all possible transient errors.

Data collected by the Autodiagnostics from May 1977 through April 1978 is presented in Table 3a, Table 3b, and plotted in Figure 11. The concept of module time is employed here. Module Time, t_m, is the sum of the times of operation of individual modules in a set of identical modules. The reliability experience that it represents is assumed to be the same as that which would be gained by observing the renewal process of a representative module of the set for a length of time t_m. The Mean Time Between Errors (MTBE) is calculated by dividing the module hours by the number of errors. For the pieces of equipment under consideration, this corresponds to the errors that occured during the time the module was exercised. The CPU was diagnosed approximately 10% of the time by the combination of the instruction and trap diagnostics. The rest of the time being equally divided between the memory and the Slocal. The overall MTBE is calculated using the total time and errors for that period (Table 3 and Figure 11). The final method uses a two month sliding "Window" to smooth out the measurements. Due to the level of resolution in the detection of transient errors by the diagnostics these MTBE's are for the diagnostics as opposed to the hardware.

As may be noted from Figure 11 there has been a general improvement in the operation of Cm* by a factor of about 10:1 over a period of a year. The most noticeable change in the MTBE occurred in January 1978. In seeking to explain this sudden change we observed that it was during that month that slight modifications were introduced into the operational routine observed by the Autodiagnostics. Diagnostic programs are loaded into an idle Cm from secondary storage, in this case a DEC-Tape mounted on the Host. In the past the typical elapsed time between loadings of two diagnostics in the sequence was set at approximately 7 minutes. This caused undue mechanical wear on the DEC-Tape drive and the interval was reset to approximately 30 minutes in order to avoid this. It is commonly noticed in general computing practice that the frequency of "crashes" is closely correlated to the load on the computing system. In particular, heavy I/O traffic seems to be responsible for transient errors. If this is in fact true for Cm* then, conceivably, the less frequent loading of diagnostics into idle Cms may be responsible for the less frequent occurrence of transient errors noted by the Autodiagnostics after January 1978. In an experiment now under way the interval between diagnostic loadings has been reverted to 7 minutes. If the hypothesis is valid the MTBE should drop back to its earlier value of about 100 hours from its present one of about 500 hours.

Month	Mod-Hrs	Occurrences of Transient Error Events Memory	Instr.	Trap	Slocal	Total
May77	841.8	13	4	4	3	24
June	888.1	6	2	2	10	20
July	931.4	2	0	1	7	10
Aug.	652.4	0	0	0	0	0
Sep.	674.0	5	0	1	6	12
Oct.	2091.2	6	0	0	13	19
Nov.	549.1	3	0	0	5	8
Dec.	1134.6	5	1	2	4	12
Jan.78	2395.8	2	0	0	1	3
Feb.	1926.6	1	2	0	0	3
March	662.9	0	0	0	0	0
April	2328.2	1	2	0	4	7
Total	1.72Yrs	44	11	10	53	118

(Total 15,072.1 Module Hours of Operation on Cm*)

Table 3a Transient Error on Cm* - Occurrences

		Mean Time Between Error (MTBE) On Cm*					
Month	Mod-Hrs	uComp	CPU	Memory	Slocal	Cm	Window
May77	841.8	22.1	17.5	27.0	116.9	35.1	-
June	888.1	48.4	37.0	71.7	55.4	44.4	39.3
July	931.4	170.8	155.2	194.0	55.4	93.1	60.7
Aug.	652.4	358.8	65.2	293.6	293.6	652.4	158.4
Sep.	670.0	61.8	77.4	60.7	75.8	67.4	132.6
Oct.	2091.2	191.7	209.1	156.8	85.5	123.0	102.4
Nov.	549.1	119.9	82.2	92.5	37.9	68.6	105.6
Dec.	1134.6	92.3	93.8	91.4	99.9	94.6	84.2
Jan.78	2395.8	1146.4	1200.3	546.3	342.6	798.6	235.4
Feb.	1926.6	556.8	571.2	527.9	256.2	642.2	720.4
March	662.9	597.0	405.0	192.2	65.6	662.9	863.2
April	2328.2	655.0	660.0	645.0	90.8	332.6	427.3

(Overall MTBE=127.7 Mod-Hr, @15,072.1 total Module-Hours)
(Overall MTBE=562.6 Mod-Hr, @ 7,313.5 since January 1978)

Module Hour = one hour of testing on one Computer Module
uComputer = CPU + Memory + Peripherals
CPU = Instruction test + Interrupt & Trap test
Memory = standard DEC DZKMA MOS memory diagnostic program
Slocal = CMU-built local Switch for address mapping
Cm = uComputer + Slocal
Window = average of previous and current month

Table 3b Transient Error on Cm* - Mean Time Between Error (MTBE)

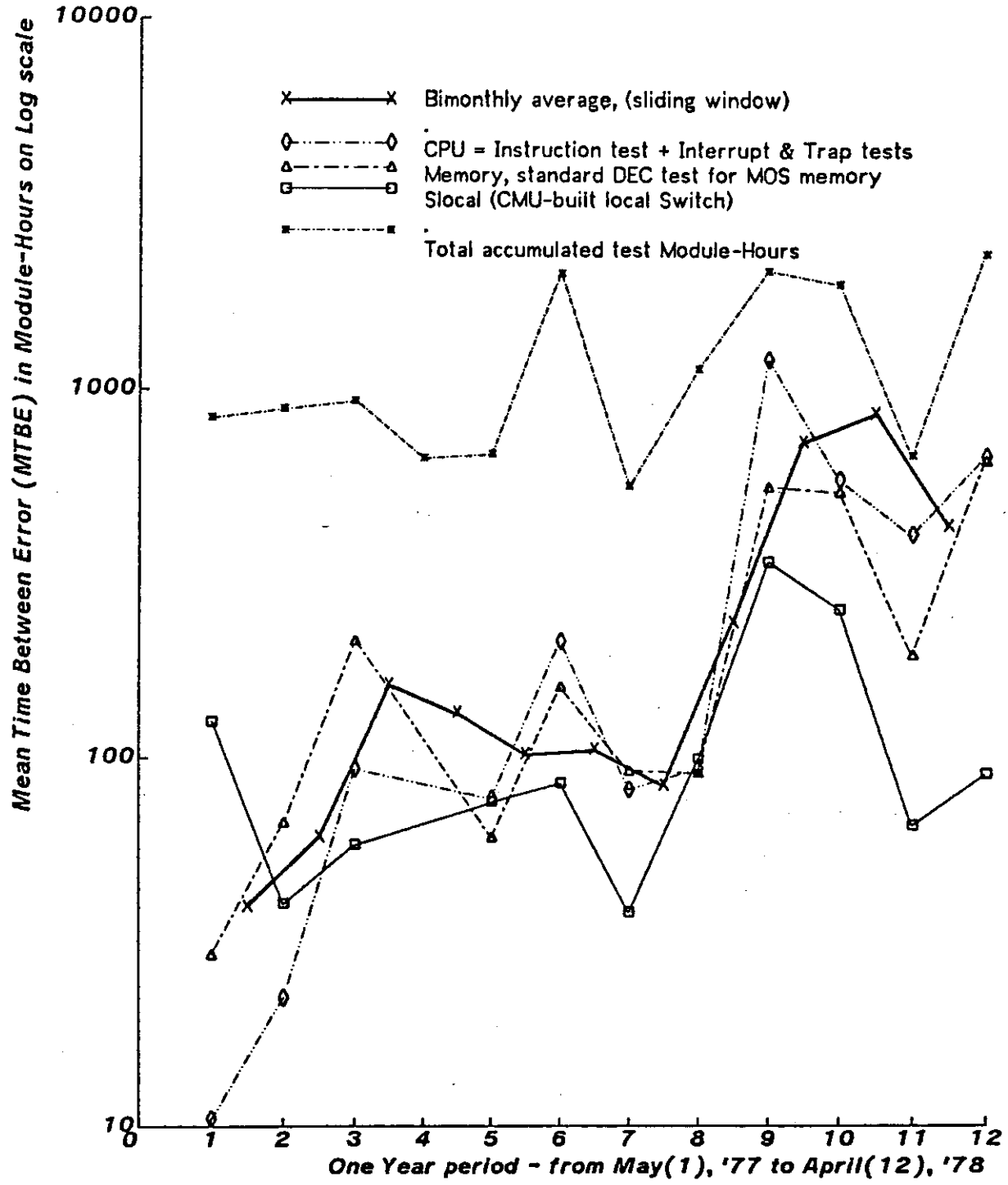


Figure 11. Mean Time Between Errors (MTBE) on Cm*

Mode	Diagnostic test used: Memory	Instr.	Trap	Slocal	Total	%Total
Burst	5	1	1	13	20	31.2
Simult	5	1	0	5	11	17.2
Isolat.	13	3	2	15	33	51.6
Total	23	5	3	34	64	100.0
%Total	35.9	7.8	4.7	51.6		

(Data collected from September, 1977 through April, 1978)
(MTBE = 183.7 Mod-Hrs, @11,758.4 total Mod-Hrs)

Table 4 Distribution of Transient Error on Cm* by mode of occurence

Three basic patterns were noticed in the transient errors: multiple errors occuring together in the same Cm (Burst); simultaneous errors occuring on different Cms (Simultaneous); and finally isolated errors (Isolated). Table 4 contains the data broken down into these classes. These data were collected between September 1977 to April 1978 on the Cm* system.

The most common cause of the "Burst" type of error is the diagnostic program being destroyed. This can be seen in either spurious halts or continuous reporting of an error. Bursts may also be caused by transient errors of a duration which is long compared to the time resolution of the diagnostic. The destruction of the diagnostic program due to errors in transmission during the loading process is not very likely since all such transfers are checksummed.

The simultaneous case is where two different Cm's are affected by an error within the same period of time. The final case is that of isolated errors which basic redundancy techniques would be able to tolerate.

From the data in Table 3b, representing about 1.7 module-years, one would expect a transient error about every 130 hours (or over 500 hours since January 1978) in a typical computer Module with 48K bytes of dynamic MOS memory. This number should be considered to be an upper bound, since not all transients will be caught by repeated execution of a diagnostic program. It is also interesting to contrast the transient failure rate to the hard failure rate of the previous section. A comparison suggests a ratio of about 100:1 overall for transients over hard failures (30:1 since January).

For each occurence of a transient error, as indicated in Table 4, there is a 31% probability that the internal state of a processor will be destroyed and reinitialization is required (Burst). When a transient occurs, a simple dual processor would fail 17% of the time. This is indicated by the fact that 17% of all transient errors were detected in more than one processor simultaneously. Redundancy techniques would tolerate 52% of all transient errors, since these isolated errors did not destroy the test program.

4. C.vmp: A Voted Multiprocessor

4.1. Design Goals

A design study was initiated in the summer of 1975 to examine fault tolerant architectures in industrial environments. Major attributes of this environment were electromagnetic noise, less knowledgeable users, and nonstop operation. From these attributes the following design goals were established:

1) Permanent and transient fault survival

The system should have the capability to continue correct operation in the presence of a permanent hardware failure--i.e. a component or subsystem failure--and in the presence of transient errors--i.e. a component or subsystem is lost for a period of time due to the superposition of noise on the correct signal.

2) Software transparency to the user

The user should not know that he is programming a fault tolerant computer, with all fault tolerance being achieved in the hardware. This would allow the user to rely on established software libraries, increasing the reliability of the software itself.

3) Capable of real time operation

A fault should be detected and corrected within a short period from the time the fault actually occurs.

4) Modular design to reduce down time

The hardware should be able to operate without certain sections activated. Hence, maintenance could be performed without having to halt the machine. Modularity includes the design of separate power distribution networks to be able to deactivate selected sections of the machine. The use of modules in the design also has the virtue of allowing the user to upgrade from a non-redundant, to a fully fault tolerant computer, in steps.

5) Off-the-shelf components

To decrease the amount of custom designed hardware, to be able to rely on an established software library, and to allow systematic upgrading to a fault tolerant system, the computer should primarily employ off-the-shelf components. Further, as illustrated in a companion paper [1], advantage can be taken of the greater reliability of high production volume components.

6) Dynamic performance/reliability tradeoffs

The fault tolerant computer should have the capability, under operator or program control, to dynamically trade performance for reliability.

4.2. System Architecture

Actual System Configuration

To be consistent with the design goals of modularity and software transparency, bus level voting was selected as the major fault tolerance mechanism. (See [28] for a more detailed discussion leading up to the selection of voting.) That is, voting occurs every time the processors access the bus to either send or retrieve information. There are three processor-memory pairs, each pair connected via a bus as depicted in Figure 12. A more precise definition of C.vmp (for Computer, Voted MultiProcessor) would therefore be: a multiprocessor system capable of fault tolerant operation. C.vmp is in fact composed of three separate machines capable of operating in independent mode executing three separate programs. Under the control of an external event or under the control of one of the processors, C.vmp can synchronize its redundant hardware, and start executing the critical section of code.

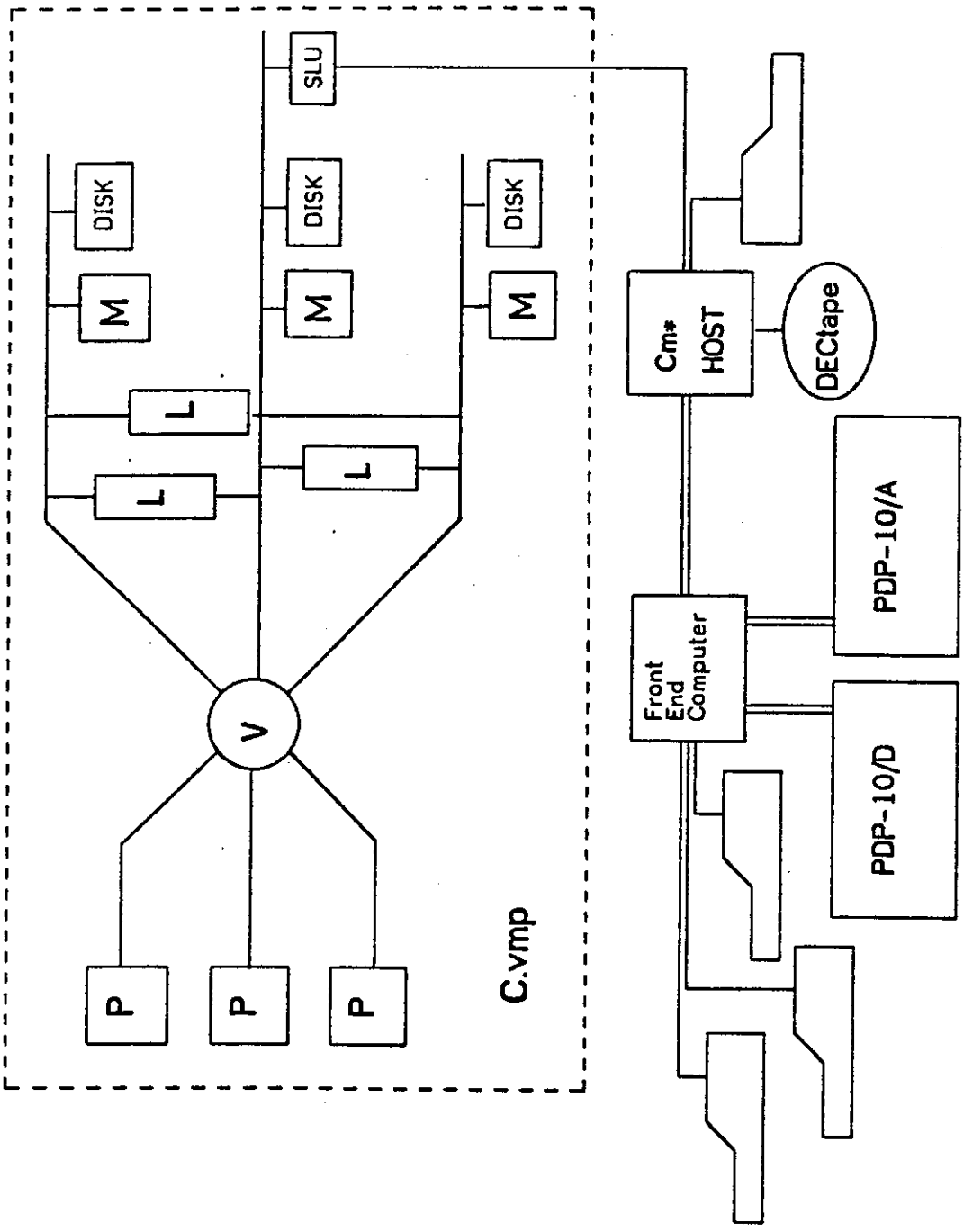


Figure 12. C.vmp configuration and connection to C-MU facilities

With the voter active, the three buses are voted upon and the result of the vote

is sent out. Any disagreements among the processors will, therefore, not propagate to the memories and vice versa. Since voting is a simple act of comparison, the voter is memoryless. Disagreements are caught and corrected before they have a chance to propagate. The nonredundant portion of the voter does not represent a system reliability "bottleneck", as will be shown later. However the voter may be totally triplicated if desired. With voter triplication even the voter can have either a transient or a hard failure and the computer will remain operational. In addition, provided that the processor is the only device capable of becoming bus master[1], only one bidirectional voter is needed regardless of how much memory or how many I/O modules are on the bus. Voting is done in parallel on a bit by bit basis. A computer can have a failure on a certain bit in one bus, and, provided that the other two buses have the correct information for that bit, operation will continue. There are cases, therefore, where failures in all three buses can occur simultaneously and the computer would still be functioning correctly.

Bus level voting[2] works only if information passes through the voter. Usually the processor registers reside on the processor board and so do not get voted upon. The PDP11, for example has six general purpose registers, one stack pointer, and one program counter. However, after tracing over 5.3 million instructions over 41 programs written by five different programmers and using five different compilers, the following average program behavior was discovered [31]:

> On the average a register gets loaded or stored to memory every 24 instructions.
>
> A subroutine call is executed, on the average, every 40 instructions, thus saving the program counter on the stack.
>
> The only register that normally is not saved or written into is the stack pointer. To maintain fault tolerance the system must periodically save and reload the pointer.

---

[1] Note that this restriction prohibits the use of "Direct Memory Access" (DMA) devices. If such devices were only allowed to communicate with the processors and the memory (not other I/O devices), a second voter between the memory and the I/O devices on the bus would be sufficient to retain fault tolerance.

[2] This bus level voting scheme can be contrasted with the Draper Laboratory Symmetric Fault Tolerant Multiprocessor [29]. In SFTMP, memory and processor triads are interconnected by a triplicated serial bus. Program tasks are read from a memory triad into local memory in a processor triad where execution takes place. After execution the results are transferred back to memory triads. The major architectural differences from C.vmp are: Serial bus rather than parallel bus, thus degrading performance. Voting only takes place on transfers from and to memory triads. Errors in the processors may accumulate to the point that their results are not comparable. Programmer has to partition problems into tasks and provide for transfer to processor triads. SFTMP has up to 14 processors that can be dynamically assigned to four triads (two are spares). When a processor fails it can be replaced in its triad by another processor. However, processors cannot operate independent of triads to improve throughput. Another voting design is described by [30]. The described system is base on an Intel 8080 microprocessor and has an output address and data bus and an input (from memory to processor) data bus. The major difference from C.vmp is that only a unidirectional voter is employed, on the input data bus. Thus only information flow from memory to processor is voted upon. There is no consideration of I/O, apart from an assertion that each I/O device on the bus requires a separate voter.

Thus normal program behavior can be counted on to keep the registers circulating through the voter.

To present a detailed description of the voter a brief digression to explain the DEC LSI-11 Qbus is necessary [32]. The 36-signal bus uses a hybrid of synchronous and asynchronous protocols.

Every bus cycle begins synchronously with the processor placing an address on the time multiplexed Data/Address Lines (DAL).

SYNC goes high and all the devices on the bus latch the address from the DAL lines. The address is then removed by the processor. This terminates the synchronous portion of the bus cycle.

In the event of an input cycle (DATI shown in Figure 13) the processor activates DIN on the bus.

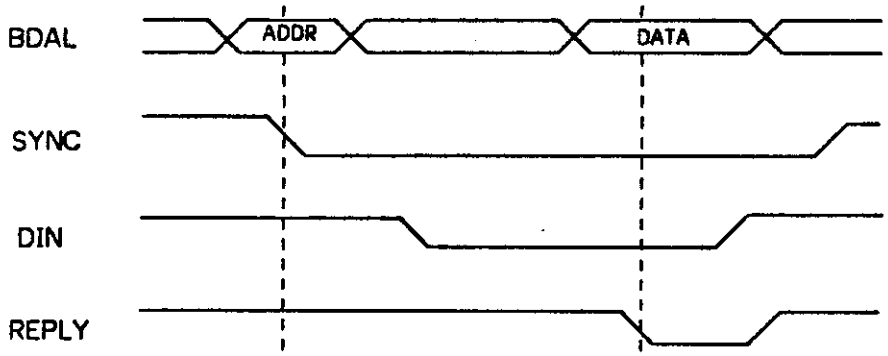


Figure 13. DATI cycle for LSI-11 computer

The addressed slave responds by placing a data word on the DAL lines and asserting REPLY.

The processor latches the data word and terminates DIN and SYNC.

In the event of an output cycle (DATO), after removing the address the processor places a data word on the bus and activates DOUT.

When the slave device has read the word it activates REPLY.

The processor responds by terminating DOUT and SYNC.

Voter modes of operation

The multiplexed paths through the voter are shown in Figure 14. Figure 14a shows the case for the (unidirectional) control lines. Signals generated by the

processor are routed from bus receivers to multiplexors which allow either signals from all three buses, or signals only from bus A, to pass to the voting circuit. The output of the voting circuit always feeds a bus driver on external bus A, but is multiplexed with the initially received signals on buses B and C. This arrangement allows all three processor signals to be voted on and sent to all three external buses; the signal from only processor A to be "broadcast" to all three external buses; and the independent processor signals to be sent to the separate external buses, albeit with extra delay on bus A:

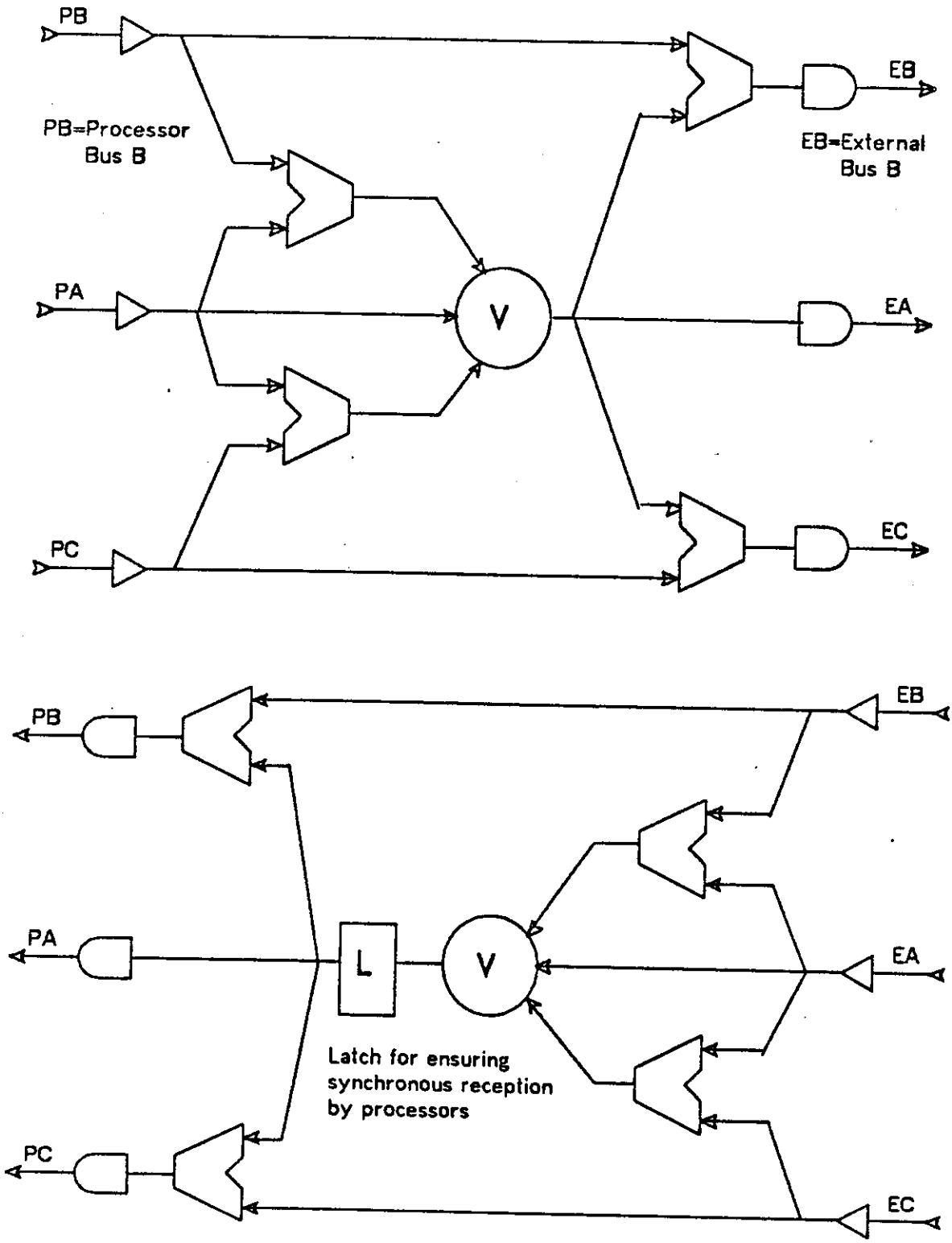


Figure 14a. C.vmp Unidirectional Voter Multiplexing

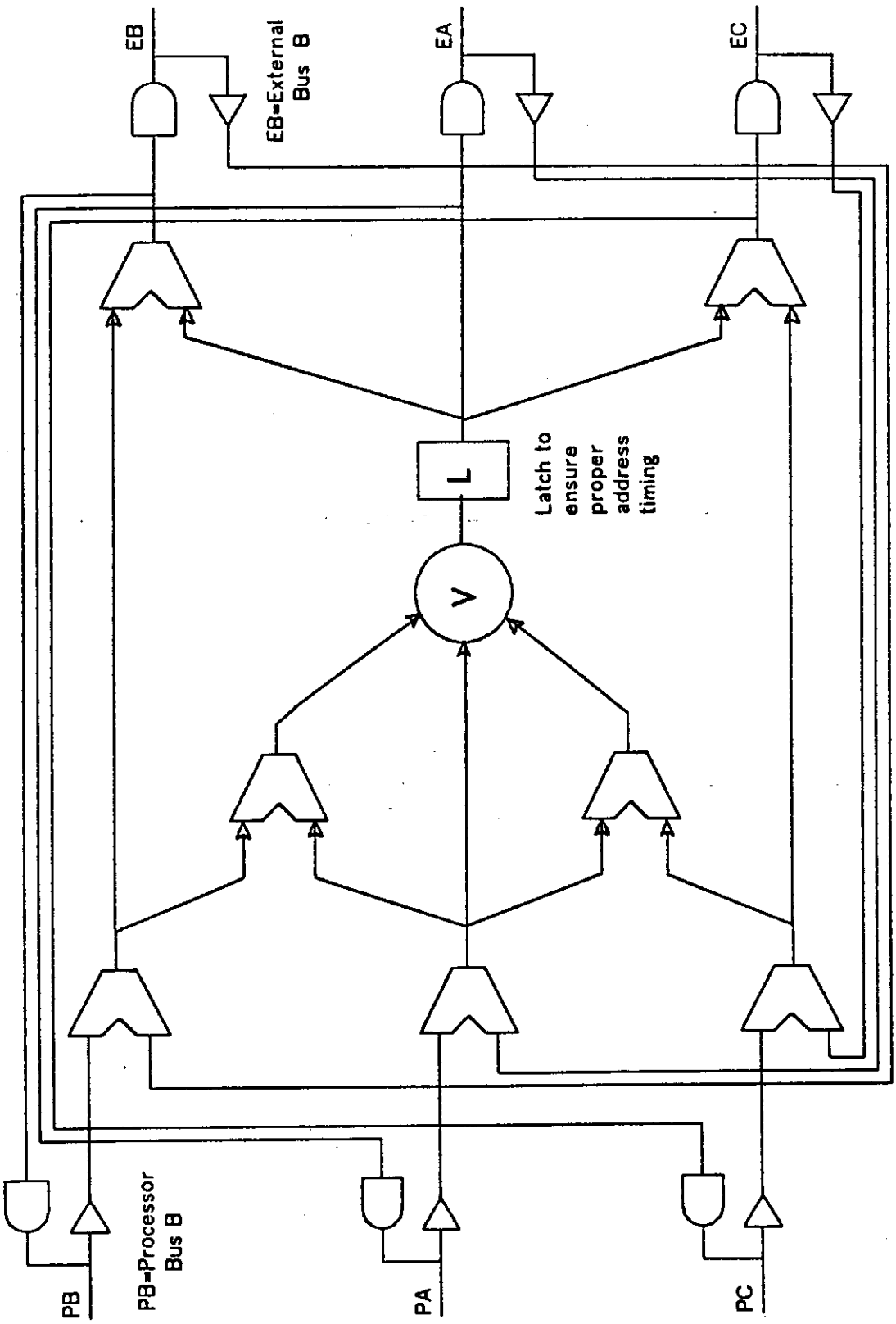


Figure 14b. C.vmp Bidirectional Voter Multiplexing

Voting mode. The transmitting portion of each of the three buses is routed into the voter, and the result of the vote is then routed out to the receiving portion of all three buses. In addition to the voting elements the voter has a set of disagreement detectors. These detectors, one for each bus, activate whenever that bus has "lost" a vote. By monitoring these disagreement detectors, one can learn about the kinds of failures the machine is having.

Broadcast mode. Only the transmitting portion of bus A is sampled, and its contents are broadcast to the receiving portions of all three buses. This mode of operation allows selective triplication and non-triplication of I/O devices, depending on the particular requirements of the user. The voter has no idea which devices are triplicated and which are not. The only requirement is that all non-triplicated devices be placed on bus A. To handle non-triplicated devices two extra lines are added to bus A. One is a special copy of REPLY for use by non-triplicated devices instead of the standard bus A REPLY, and the other is a special copy of the Interrupt Request line (IRQ).

Independent mode. Buses B and C are routed around the voting hardware. Bus A is routed to feed its signals to all three inputs of the voting elements. In this mode C.vmp is a loosely coupled multiprocessor. Switching between independent and voting modes allows the user to perform a performance/reliability tradeoff.

The unidirectional control signals generated by devices on the external buses are handled the same way as processor signals, except that the direction (external-processor) has been changed.

Figure 14b shows the more complex case of the bidirectional data/address lines. Two sets of bus transceivers replace the sets of receivers and transmitters used before, and another level of multiplexing has been added. The received signals from both sets of transceivers are fed into a set of multiplexors that choose which direction the signals are flowing. After passing through the set of multiplexors and the voter circuit, the voted signal goes through a latch which ensures that bus timimg specifications are met. From there the signals pass onto the opposite bus from which they were initially received. (Note that the drivers on the receiving bus are disabled to avoid both sinking and sourcing the same signal.)

Peripheral Devices

In most cases, triplicating a device just means plugging standard boards into the backplane, as is the case with memory. In some cases, however, the solution is not quite so simple. An example of a device that has to be somewhat modified is the RX01 floppy disk drive. The three floppys run asynchronously. Therefore there can be as much as a 360 degree phase difference in the diskettes. Since the information does not arrive under the read heads of the three floppys simultaneously, the obvious solution to this problem is to construct a buffer whose size is large enough to accomodate the size of the sectors being transferred. A disk read READ operation would then occur as follows [33].

The track and sector number to be read are loaded into the three interfaces and the "READ" command is issued.

The three floppys load their respective buffers asynchronously.

The processors wait until the three buffers are loaded and then synchronously empty the buffers into memory.

A write operation would be executed in a similar fashion.

The main synchronization problem is to find out when all three floppys have completed their task or when one of the floppys is so out of specification that it can be considered failed. Once this is determined the "DONE" signals are transmitted to the three buses simultaneously.

When in independent mode, the three processors must be able to communicate to each other. For this reason there are three full duplex single word transfer fully interlocked parallel interfaces in the system (labeled L in Figure 12). These interfaces provide data transfer between the separate processors (in independent mode) at rates up to 180K bytes per second [32]. These interfaces are used for software synchronization of the processors prior to reestablishment of voting mode, in addition to straight data transfers.

4.3. Issues of Processor Synchronization

Dynamic Voting Control

A major goal in the design of C.vmp was to allow dynamic tradeoff between reliability and performance. Ideally, when reliability is of less importance, the machine should be able to split into a loosely coupled multiprocessor capable of much greater performance. Conversely, when reliability becomes crucial, the three processors ought to be able to resynchronize themselves and resume voting. Consideration of dynamic voting mode control led to the following features:

In transitioning from voting to independent mode, a simple change in the multiplexing control signals causes the next instruction to be fetched and executed independently by the three processors;

In order to ensure proper synchronization of all processors in transitioning from independent to voting mode, a delayed transition forces an interrupt, presumably after each processor has had ample time to execute a "WAIT" instruction. ("WAIT" halts the processor until an interrupt occurs.)

Two bits are provided in the voter control register for voter mode control. The first, a read-only bit, monitors the state, returning "0" if voting, and "1" if not. The other, a read/write bit, chooses the desired mode. Each processor has a copy of the voter control register, and a vote is taken on the mode control bit. This control register is

accessed like any I/O device register, as a specific memory location (in this case, 167770).

Dynamic voting mode control has been demonstrated by a test program. When in voting mode, setting the appropriate bit in the control register causes the three processors to split apart and begin executing separately. To resynchronize the processors, a simple handshaking protocol is used, in which each processor waits for both of the others to signal permission before clearing the control bit. (A more sophisticated protocol would provide for a timeout if one of the processors has failed, with efforts to recover from such a situation.) After clearing its copy of the control bit, each processor releases control of its bus and ceases execution via a "WAIT" instruction. The ensuing interrupt generated by the voter then serves to resynchronize the three processors, and the first instruction of the interrupt service routine is the first instruction executed in voting (fault tolerant) mode.

Bus Control Signal Synchronization

There are two levels of synchronization used in C.vmp to keep the three processors in step: bus signal synchronization and processor clock synchronization. The first type of synchronization deals with the bus control signals. The voter uses RPLY to synchronize the three buses, as it is asserted by an external device (memory and I/O devices) once every bus cycle. Thus, processors can stay in step if they receive RPLY concurrently. A set of possible voting circuits is shown in Figure 15. (The boxes labeled "V" are voters, and the boxes labeled "T" are delays.) The first voter is the one used for the data/address lines. The other voters attempt to maintain synchronization of five critical control lines (SYNC, DIN, DOUT, IAK, and RPLY)[3] by waiting an appropriate period of time for a lagging control signal. (The delay is not only selected long enough that a lagging device is far enough out of specification to be suspect, but also short enough not to degrade performance severely. For maintaining processor synchronization, a value for "T" of at least one microcycle--400 nsec--is desirable, as processors are most likely to slip just one microcycle in the five to ten microcycles between bus cycles rather than to become several microcycles out of synchronization.)

[3] SYNC is used to clock the address lines, and is left asserted for the remainder of the bus cycle; DIN indicates a read cycle; DOUT indicates a write cycle; IAK is used to acknowledge receipt of an interrupt request; and RPLY is asserted to indicate that the device has responded to the request indicated by the the previous four signals.

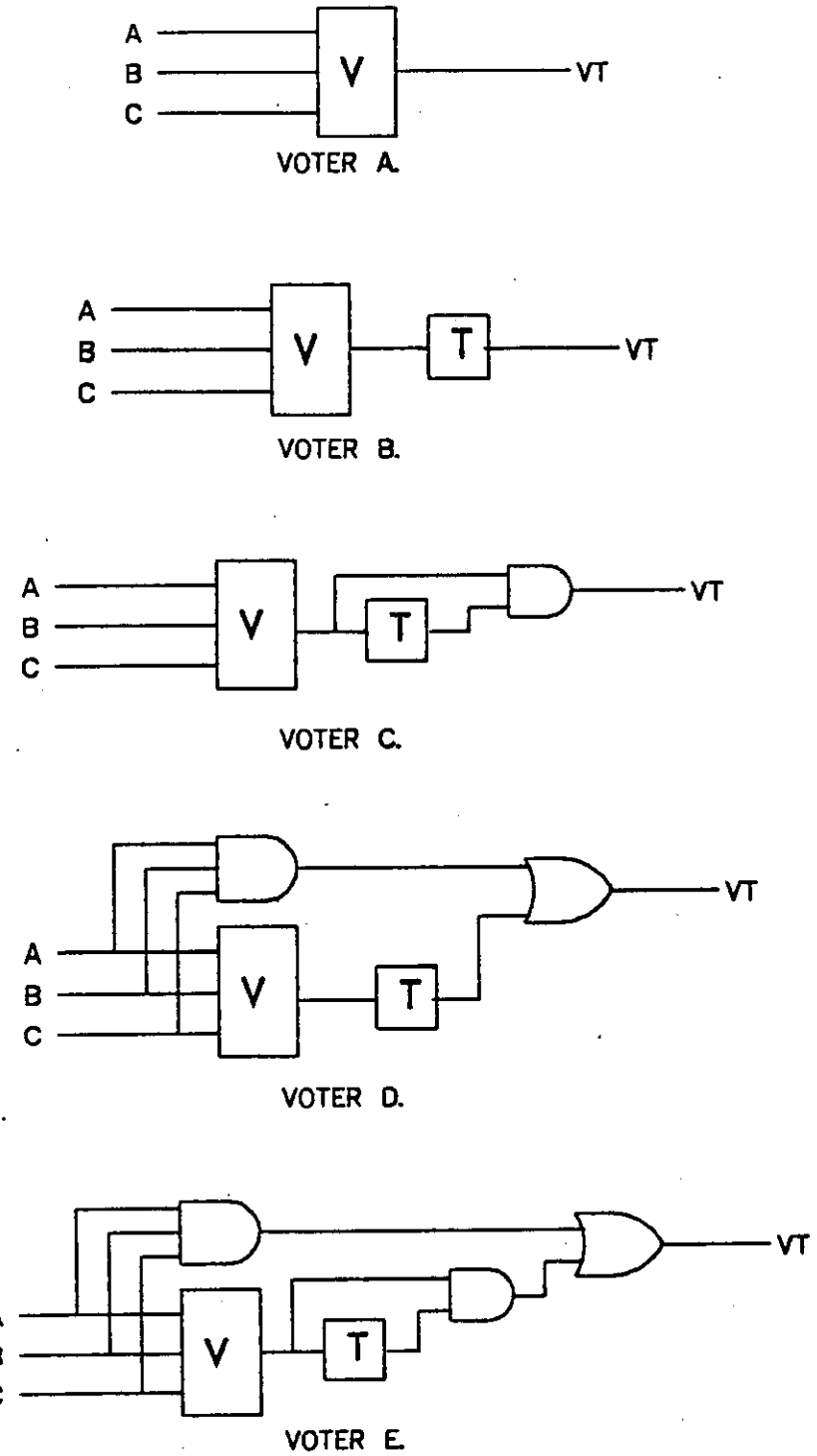


Figure 15. Synchronizing Voter Circuits

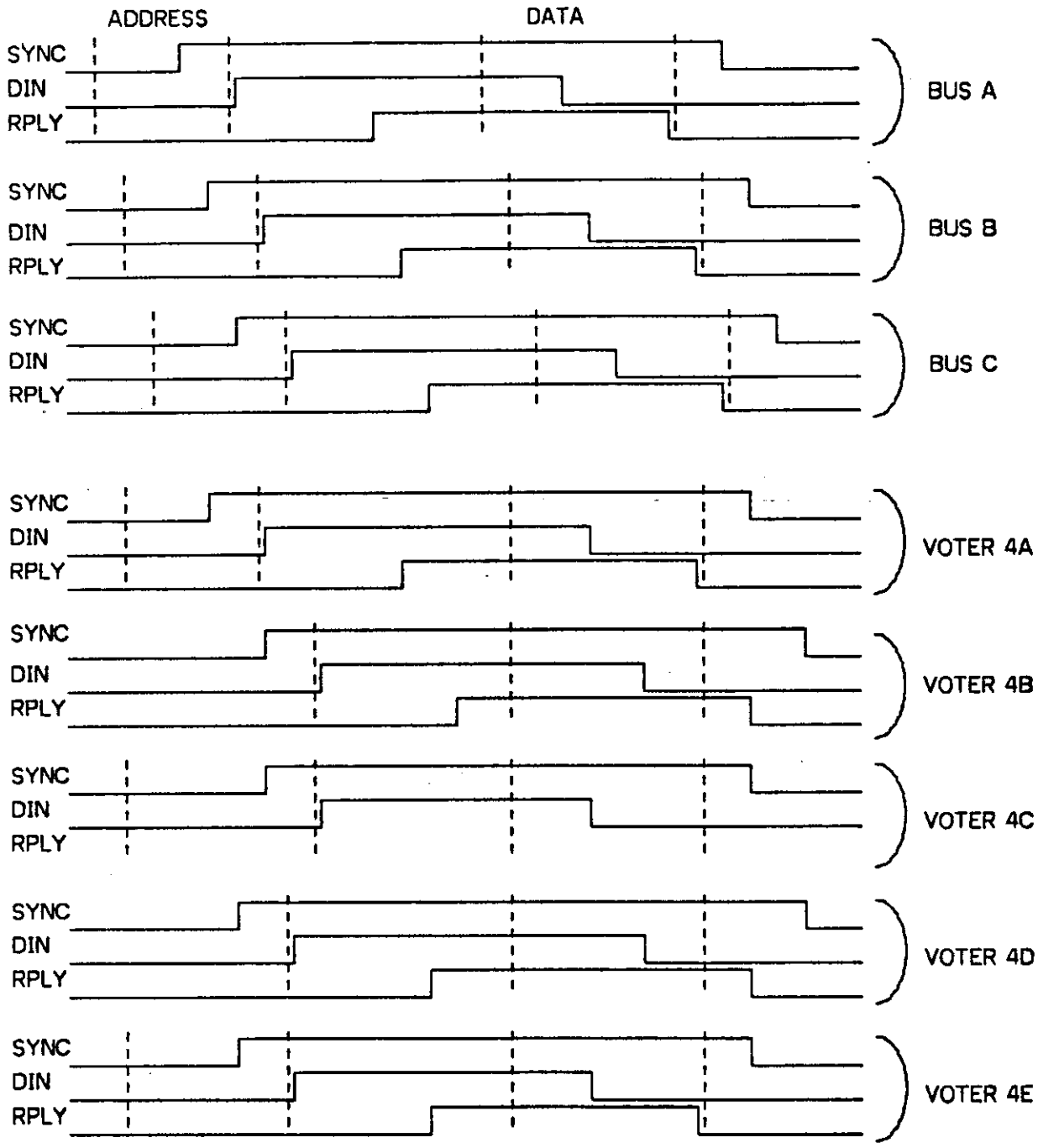


Figure 16. DATI bus cycle with desynchronized processors

The first circuit considered for synchronizing the five control lines was voter

15a. This was rejected because it provides no synchronization at all: if a signal fails high, the voter passes the first of the other two to be asserted without regard to the second. Thus, if the two remaining processors get at all out of step, the voting process fails.

The second circuit, voter 15b, provides a measure of synchronization by waiting a time "T" for the third signal after two have been asserted. However, performance is degraded because this delay occurs even when all three processors are working and synchronized. Also, control signals will continue to be asserted after they should be in relation to the data on the bus, failing to meet bus specifications. (RPLY is asserted after DATA is invalid, see Figure 16.)

The third circuit, voter 15c, fixes the problem of meeting bus specifications by having a slow-rising, fast-falling delay after the voter. However, performance is still degraded by the presence of the delay even when all is well.

The fourth circuit, voter 15d, addressed the performance problem by providing a second path through the voter for when all three processors are working. However, the delay used after the voter to provide synchronization still causes the signal to fail bus specifications, and also causes some amount of unavoidable performance degradation. (RPLY is asserted after DATA is invalid, see Figure 16.)

The last circuit, and the one used (voter 15e), combines the features of the previous two. Thus, a slow-rising, fast-falling delay is used in order to meet bus specifications; and a second path through the voter is provided for optimal performance when all is well. Note that the fast-falling feature of the delay not only allows bus specifications to be met, but also removes any performance degradation due to the voting process when all three signals are in step. This circuit was used for SYNC, DIN, DOUT, IAK, and RPLY in C.vmp. The value for "T" is about 400-500 nsec for SYNC, DIN, DOUT, and IAK, and about 75-100 nsec for RPLY. This method allows the three processors to receive RPLY within five nanoseconds of each other, and thus to stay synchronized.

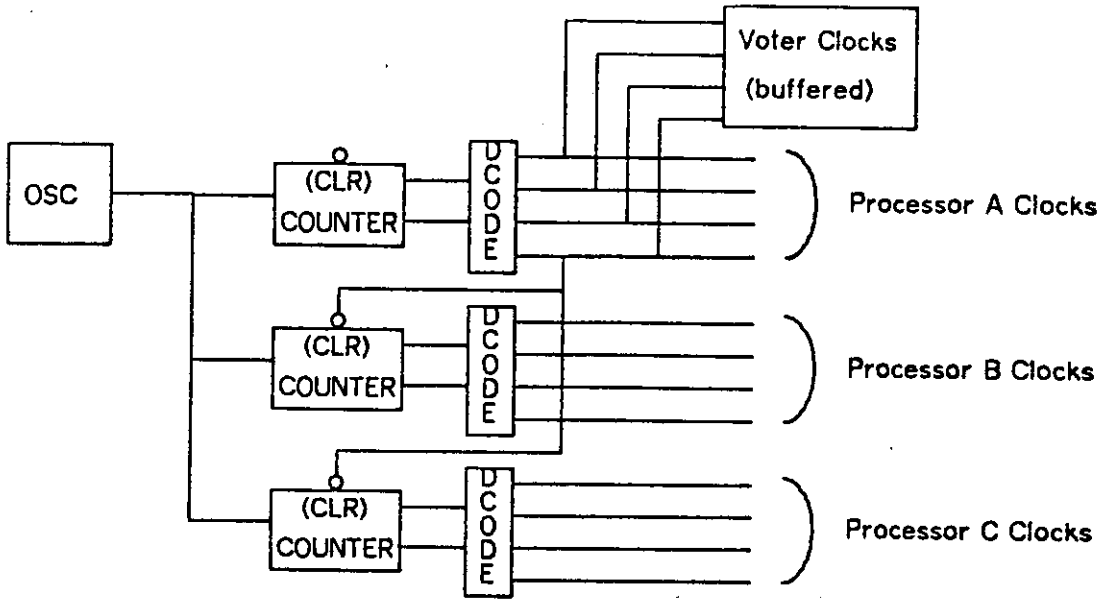


Figure 17a. Original processor clock synchronization

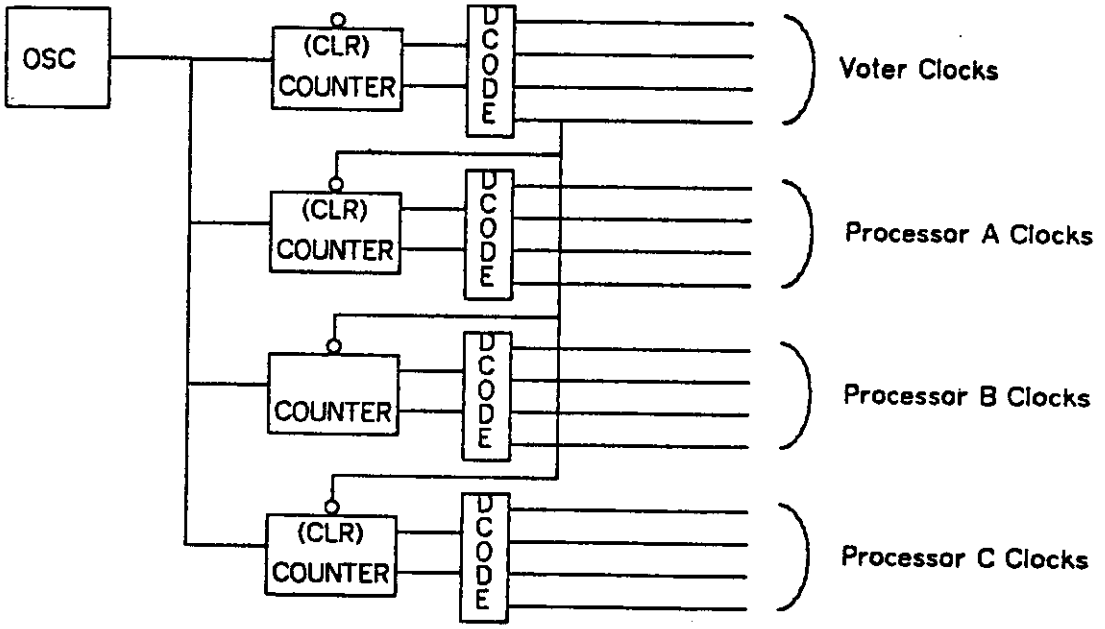


Figure 17b. Current processor clock synchronization

System Clock

Perhaps the most critical timing problem encountered in the design of C.vmp was the synchronization of the four phase processor clocks, and also the memory refresh[4] timing oscillators. This part of the design was left untriplicated in C.vmp due to its very small size, hence high reliability, relative to the rest of the machine. The original design, shown in Figure 17a, used the oscillators on processor A to drive the clock circuits on all three processors, and the decoded clock signals of processor A to feed the voter and to synchronize the phases of the other two processors by forcing phase one when processor A was in phase one. This original design worked fairly well, as processors B and C were closely synchronized, but the extra loading placed on the clocks of processor A caused them to lag several nanoseconds behind, a significant figure for pulses of less than 100 nsec duration. This resulted in sufficient unreliability that the mean time between crashes in voting mode was never more than five minutes. Therefore, a new clock circuit, shown in Figure 17b, was installed in the voter to drive and synchronize the processor clocks. All three processors were wired exactly the same, needing only three wires changed on each board. Since this change was made, the mean time between software discernable disagreement has been over 250 hours, with one run of more than 900 hours before crashing.

Initial measurements using the disagreement detection circuit attached to all the bus control lines showed no errors on any of the three buses over periods ranging between eight to forty hours. (Note that data/address lines were not included.) This indicates that the processors are well synchronized by the current design.

4.4. Performance Measurements

Processor Execution/Memory Fetch Time

An important parameter in the design of fault tolerant computers is the amount of performance degradation suffered to obtain greater reliability. In a triplicated architecture such as C.vmp, the obvious loss of two-thirds of the available computing power is unavoidable. This was the reason why C.vmp was made flexible enough to switch between voting (fault tolerant) mode and independent (high performance) mode. However, this fundamental loss due to triplication is not the only loss: the voter cutting and buffering all the bus lines introduces delays of 80 to 140 nsec in the signals between the processors and the memories.

Because the LSI-11 is a clocked machine, these delays are not too significant in and of themselves. However, the latching of RPLY from slave devices on the external buses in order to preserve processor synchronization turns out to be the more dominant degradation factor. The voter latches RPLY one clock phase (100 nsec) before the processors to allow sufficient latch settling time for minimizing the

---

[4] Note that the LSI-11 uses dynamic MOS RAM memory, which requires continual refreshing. This is normally done by processor microcode at regular intervals of about 1.67 milliseconds.

probability of a runt pulse [34]. The delays in the control lines due to the voter cause the external RPLY to return during the phase on which the processors sample RPLY but after the voted RPLY has already been latched. Thus, the voted processors must wait one extra clock cycle (four phases/400 nsec) to receive their RPLY after asserting SYNC than would a nonredundant LSI-11. The same sort of delay happens on the falling edge of RPLY, causing up to two clock cycles to be lost in one complete bus cycle. These losses could likely be prevented by more careful selection of timing components within the voter, and more importantly, by choosing different timing on the memory boards.

Measurements were taken on the various bus cycles to learn what amount of degradation actually was occurring. These measurements, and all others presented later, were taken on the voted processor (C.vmp) and on either processor B (PBB) or C (PCC) in independent mode. (Note that in independent mode, bus A passes through the entire voter via the broadcast multiplexing, while both buses B and C pass only through a bus receiver/driver pair. Comparison tests with other LSI-11's showed that processors B and C operated fully as fast in independent mode as a standard LSI-11.) The degradation within bus cycles introduced by the voter ranges from 27% to 67%, with 40% degradation for the most common (read) cycles.

As the LSI-11 does not saturate its bus, the above figures are worse than the overall processor degradation. A second step in measuring degradation was to check the different phases of instruction execution. Tests were made using the MOV, TST, and BR instructions[5] as typical double operand, single operand, and zero operand instructions. From this data, a prediction can be made of performance degradation by using instruction frequency data provided by [35]. Table 5 summarizes the calculations, showing that the voting process should degrade instruction execution performance by roughly 14%.

phase	C.vmp	PCC	C.vmp/PCC
fetch	7.00	6.00	1.167
source	2.69	2.09	1.287
destination	3.68	3.22	1.143
execution	3.53	3.53	1.000
total	16.90	14.84	1.139
time (usec)	6.760	5.936	

TABLE 5. Normalized Instruction Phases

The third stage for measuring performance was to run a set of test programs with representative mixes of instructions and addressing modes to test the validity of

[5] MOV loads the destination from the source, TST examines the destination for various conditions, and BR causes an unconditional transfer of control.

the above model. Table 6 compares the triplicated processor with a single LSI-11, both without faults and with certain induced faults. These faults were in the two most critical bus control signals, SYNC and RPLY, and represent worst case failures. Each signal was forced to be either always asserted (hi) or never asserted (lo) on one of the three buses.

program[6]	DVKAA	DZKMA	QSORT
(unit)	msec	min	sec
LSI-11	18.51	7:03	11.9
C.vmp (normal)	21.4	8:23	14.0
C.vmp (RPLY hi)	21.4	8:23	14.0
C.vmp (RPLY lo)	21.4	8:23	14.0
C.vmp (SYNC hi)	21.4	8:23	14.1
C.vmp (SYNC lo)	23.6	9:20	15.6
C.vmp/LSI	1.157	1.189	1.176
C.vmp'/LSI	1.324	1.276	1.311 (SYNC lo)

TABLE 6. Sample Program Execution Times

As illustrated by Table 6, a degradation in performance of about 16-19% can be expected, as compared to a standard LSI-11. This figure is somewhat larger than predicted by the above model, which can be attributed to the greater degree of degradation in such functions as memory refresh, which is done by the processor microcode (18.5%), and also to normal deviations of programs from the "standard" instruction mix.

The measurements involving the four failure modes show that only certain failures will cause further degradation: those which cause the processor's synchronizing signals (e.g., SYNC, DIN, and DOUT) never to be asserted. Even in these extreme cases, only another 12-14% slowdown is experienced. Most faults, however, would not degrade the speed at all, but just the future reliability. For instance, the loss of power to a bus would force all signals to ground, which is the active assertion level (hi) on the LSI-11 bus. Only lo failures in the five bus control signals which require synchronization will cause any degradation. (Recall that there are a total of 36 bus lines.)

Disk Access Time

The last performance measurements involved the floppy disks used for mass storage on C.vmp. Access time to a particular position on a rotating memory is

---

[6] DVKAA is the basic instruction diagnostic, testing all instructions and addressing modes. DZKMA is the memory diagnostic, and would tend to make more memory references than average. QSORT is an example of compiler-produced code, being an integer sorting program coded in BLISS-11.

assumed to be directly proportional to the initial position of the disk. Since the hardware makes no attempt to synchronize disk rotation, access to the triplicated disks will take the maximum of the three times. In general, for n disks, the access time is given by:

$$T_n = MAX\ (t_1, t_2, \ldots, t_n)$$

Assuming that each access time t is uniformly distributed over the normalized range [0,1], the expected value for access time is:

$$T_n = n / (n+1)$$

This means that for a single disk (n=1), we can expect to wait .5 rotations; for the triplicated disk (n=3), .75 rotations. This gives a 50% degradation in access time for the triplicated disks over the non-triplicated disk for random accesses. This figure was verified to an extent by experimental data. In reading 50 sectors in a random pattern from the same physical track, the triplicated machine experienced about 51% degradation, a very close confirmation. However, if the track was also chosen at random for each of the 50 sectors, the triplicated machine was only 18% slower than the single disk system. The model failed to consider that, although sector access time is affected by the diskettes being out of phase, track access time is the same regardless of triplication.

Another shortcoming of the disk performance model based only on consideration of the diskettes being out of phase with each other is the impact of the resulting slowdown on nonrandom disk access patterns. The impact of this can be much more severe (or much less severe) than predicted, depending on the pattern of nonrandom disk accesses. For instance, the RT-11 floppy disk software uses a 2:1 interleaving of sectors in order to minimize access time for sequential file storage[7]. The extra delay due to voting causes this interleaving to be insufficient for achieving much speedup in accesses, as illustrated by Figure 18. Waiting for all three drives to read a sector can cause the first two drives to overrun the next sector in sequence before the third drive has read the initial sector. This causes part of an additional revolution to be required on the next sector read. For the example shown, a nontriplicated disk drive requires only 0.375 revolutions to read sectors 1 and 3, while the triplicated drive needs 1.75 revolutions. The specific values depend on the number of sectors per revolution, the access pattern (and interleaving scheme), and the degree to which the three disks of the triplicated drive are out of phase.

---

[7] 2:1 interleaving means that only every other sector on a track is read when reading sectors sequentially. As some amount of time is necessary to read the data into memory after it has been fetched from the diskette, this allows all 26 sectors of a track to be read in just two revolutions rather than in twenty-six revolutions.

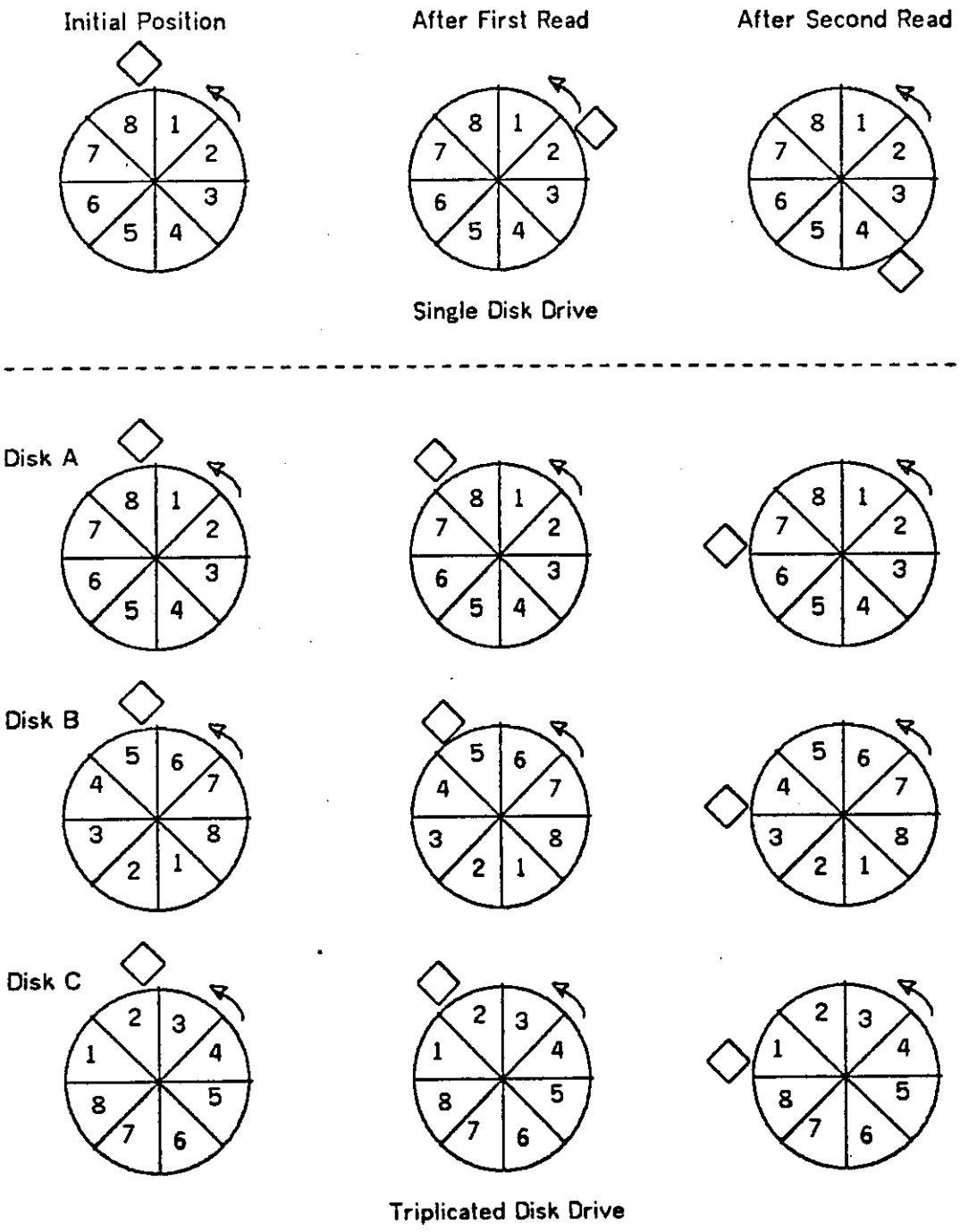


Figure 18. Effects of disk triplication on sequential access
(2:1 Interleaving)

Table 7 summarizes timing data collected by a program which was written to test different interleaving schemes. A number of consecutive logical sectors were read, which mapped into the same number of physical sectors in the pattern dictated by the desired interleaving. In addition, a test program was assembled under RT-11, using its 2:1 interleaving, to examine the impact of increased disk latency on typical operations. Figure 19 plots access time versus interleaving factor for reading 1000 sectors sequentially. The data indicates that perhaps the best sequential file access could be achieved for triplicated disks using 8:1 interleaving. The point to be made about replicated disk access time is that it is very pattern sensitive: very little degradation due to replication occurs in sequential accesses without interleaving, but great degradation is seen when interleaving is used. Instead of the factor of ten speedup available with 2:1 interleaving on a single disk, only a factor of roughly 1.5 is possible (using 8:1 interleaving) on a triplicated disk.

sectors	interleave	C.vmp	PBB	C.vmp/PBB
10	1:1	1.69	1.66	1.021
10	2:1	1.55	0.17	9.218
50	1:1	8.51	8.06	1.055
50	2:1	7.66	0.81	9.403
1000	1:1	171.2	159.9	1.071
1000	2:1	153.9	14.6	10.540
assembly	2:1	109.6	15.8	6.937

TABLE 7. Disk Timing Tests

All measurements given in Table 7 are in seconds.

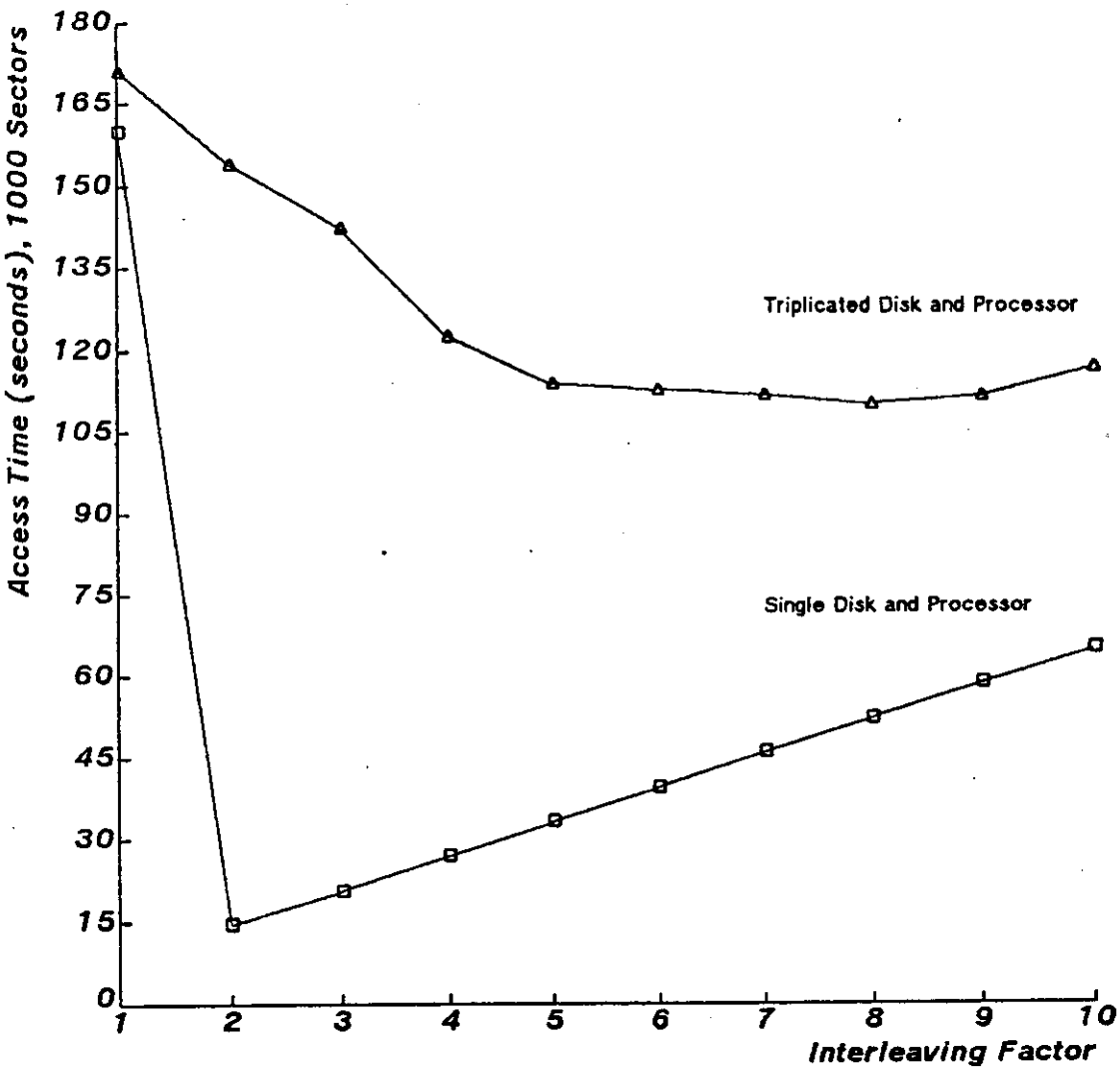


Figure 19. Disk Access Time vs. Interleaving Factor

4.5. Operational Experiences

Operating History

Implementation of C.vmp has been completed, and stable performance achieved. The software is a standard, unmodified single-user diskette-based real time operating system (RT-11). The system has been utilized under actual load conditions with students doing projects in an introductory real time programming course. The students were supplied with an RT-11 software manual and a short paper on C.vmp specific data (i.e. location of the power switches, reminder to load three diskettes, etc.). To these users, C.vmp successfully appeared as a standard LSI-11 uniprocessor running standard software.

C.vmp System Reliability

C.vmp has repeatedly demonstrated hard failure survival by bus power switching and board removal (see comments later about on-line maintenance). Another aspect of fault tolerance is transient fault survival. The only transients which should cause C.vmp to crash are those occurring simultaneously in more than one module. According to the data from Cm* presented in Section 3, such transients make up 17% of the total, occurring roughly every 1000 hours. The mean time to crash should equal or exceed this figure. Indeed, as the hardware situation has been stabilizing, C.vmp's reliability has been increasing toward this order of magnitude. Table 8 summarizes C.vmp crash data for the nine month period from August 1, 1977 to April 30, 1978. Note that software or user caused crashes have not been included in the data. Also, repeated crashes (ones due to the same cause) have been removed. Due to uncertainty as to the exact causes of many crashes, dual tables have been constructed giving the "best case" and "worst case" figures. Crashes which may have been software or user caused are included in the worst case, but not in the best case data.

month	mean	std dev	median	number	uptime
August	64.8	91.9	28.0	5	323.8
September	108.7	139.6	35.6	4	434.9
October	35.5	51.1	19.8	16	568.3
November	49.3	33.0	52.0	10	492.9
December	204.8	191.6	113.1	3	614.5
January	95.4	104.3	70.5	7	667.7
February	258.8	78.6	258.8	2	517.6
March	298.3	276.4	298.3	2	517.6
April	352.4	114.2	352.4	2	704.7
Total	96.5	167.8	30.6	51	4921.1

TABLE 8A. C.vmp Crash Data (worst case)

month	mean	std dev	median	number	uptime
August	81.0	96.1	34.6	4	323.8
September	217.4	132.4	217.4	2	434.9
October	142.1	44.5	125.7	4	568.3
November	246.5	167.3	246.5	2	492.9
December	614.5	0.0	614.5	1	614.5
January	--	--	--	0	667.7
February	517.6	0.0	517.6	1	517.6
March	--	--	--	0	596.7
April	704.7	0.0	704.7	1	704.7
Total	328.1	470.8	114.3	15	4921.1

All times given in Table 8 are in hours.
(std dev is the standard deviation.)

TABLE 8B. C.vmp Crash Data (best case)

The voter induced transient failures are mainly due to construction. The wirewrap boards used in the voter are prone to socket failures. These sockets are being systematically replaced, with a consequent improvement in mean time to crash (MTTC). With permanent construction techniques (e.g. printed circuit boards) the voter should be removed as a source of system crashes.

One measure of transient fault survival lies in the severity of the methods necessary for recovery. Five levels of recovery exist: (1) CONTINUE execution at the same location without any change to processor registers or memory; (2) RESTART the program in memory, which will also reset the I/O devices and processor registers; (3) RELOAD the program into memory, also resetting the I/O devices and processor registers; (4) RESET the processors and reload the program; and (5) DEBUG the hardware to whatever extent is required to restore stable operation. Table 9 summarizes this data in correspondence to the entries of Table 8.

month	continue	restart	reload	reset	debug
August	0	1	3	0	1
September	0	0	2	0	2
October	0	5	7	1	3
November	0	1	7	1	1
December	0	0	2	0	1
January	0	7	0	0	0
February	0	1	0	0	1
March	0	2	0	0	0
April	0	2	0	0	0
Total	0	19	21	2	9

TABLE 9A. C.vmp Crash Recovery Data (worst case)

month	continue	restart	reload	reset	debug
August	0	0	3	0	1
September	0	0	0	0	2
October	0	0	1	0	3
November	0	0	0	1	1
December	0	0	0	0	1
January	0	0	0	0	0
February	0	0	0	0	1
March	0	0	0	0	0
April	0	1	0	0	0
Total	0	1	4	1	9

TABLE 9B. C.vmp Crash Recovery Data (best case)

It is interesting to note that the majority of crashes required relatively little effort to recover from. Only a few required the processor to be actually reset, and several only required the resident monitor to be restarted. All the cases of debugging involved socket failures in the voter boards, and seem to be getting less frequent.

On-Line Maintenance

The success of the voting mechanism has been established by experiments with powering down buses and removing components, while still having the system as a whole continue operating. With a bus powered down, the associated processor and memory are, of course, lost, but the system keeps working. Defective components (if such exist) can be replaced, and the bus powered back up. Contents of the newly restored memory can be brought into agreement with the other copies by providing a read/write memory background job. Normal operation suffices to resynchronize the processor, as it starts executing code randomly until it gets in execution phase with the other two processors.

Actual experiments have included removing memory boards from one, two, or even all three buses (different 4K banks of memory from different buses). Also, a processor was removed, and the machine kept running. Even with one of the processors missing, and a different 4K bank of memory removed from each bus, the machine continued in operation.

The only problem encountered with these experiments was that restoring power to a bus sometimes causes a crash. All three buses, and even the voter itself, draw power from the same +5v supply. The transients on the power lines associated with turning on an LSI-11 processor, 12K of memory, and assorted I/O interfaces are the cause of the crashes. (These transients arise from the sudden demand for 7-10 amps of current for the various components on each bus.) Independent power supplies, as would be desireable in any case for a fault tolerant computer, are necessary to correct this problem.

The ability described above to power down selective sections of C.vmp in order to remove or replace defective modules is certainly a strength of the system as regards being a highly available machine.

The Transient Analysis Experiment

A search through the literature reveals little or no experimentation in the area of noninduced transient fault measurements. To facilitate gathering data on such effects of noise, a statistics board which straddles all the buses is under development. This statistics board latches selected information from the buses whenever a disagreement, signalling an error, is detected. This latched information is stored along with a unique time signature stamp in onboard memory for later dumping and analysis. The main experiments that we hope to perform on this machine are the following.

The first experiment consists of exposing one of the external buses to a controlled noise environment, either directly coupled through the power supply, or radiated by a noise source. The rest of the computer would be kept in a shielded environment.

With the statistics board operating, we can find out how often we get a failure, where the failure is most likely to occur, and how long a failure lasts. By repeating the experiment with different noise frequencies and different noise intensities, we can map the noise susceptibility of components in the computer. By replacing components and repeating the experiment we can determine the variation in noise susceptibility as a function of component variation due to construction.

For C.vmp to prove successful, the smallest possible correlation between a component failing and a corresponding component failing at the same time is desirable, since these correlated failures cause a system failure. In theory, we would like to prove independence between failures in similar sections of the computer. Once we know the probability of a non-fatal failure, we can expose two sections of the system that perform the same task, and record fatal failures in the system. From the first experiment we hope to compute the mean and standard deviation of a non-fatal failure. From the second experiment we hope to compute the mean and standard deviation for

a fatal failure. We can then measure the independence of two sections of the system to a noise source.

Another experiment will be in on-line maintenance through module removal with and without power switching.

5. Conclusion and Acknowledgements

Hardware and Software reliability remains high on the list of CMU's research activities. A natural evolution of our past research is an integrated hardware, firmware, software approach to systems design. Activity is being initiated to look at the next generation multi-processor which will have reliability and security as its major design goal.

Through the years many people have contributed to the concepts and implementation effort required to construct these architectures. That list might approach 100 people and is too lengthy to include here. However, we would like to acknowledge several people who have contributed to the reliability efforts. For C.mmp the list includes William Wulf, Fred Pollack, and Roy Levin. Cm* owes much to the major hardware designers Richard Swan, John Ousterhout, Kwok-Woon Lai and Andy Bechtolsheim and to the major software designers Anita Jones, Robert Chansler, Ivor Durham, Peter Feiler, and Karsten Schwans. The contributions of Mark Canepa and Steve Clark to the C.vmp design were significant. Finally, William Avery, Gordon Bell, Lloyd Dickman, Rich Olsen, Robert Swarz, Steve Teicher and Mike Titelbaum at Digital Equipment Corporation have provided information, ideas, and support critical to the success of the Cm* and C.vmp projects.

References

[1] Siewiorek, D.P., V. Kini, R. Joobbani, and H. Bellis, "A Case Study of C.mmp,C.vmp and Cm* -- II. Predicting and Calibrating Reliability of Multiprocessor Systems", this issue. [2] Bell, C. G. and A. Newell, "Computer Structures: Readings and Examples," New York: McGraw-Hill, 1971.

[3] Wulf, W. A., and Bell, C. G., "C.mmp---a multi-mini-processor", Proc. AFIPS 1972, FJCC. Vol. 41, AFIPS Press, Montvale, N.J., pp. 765-777.

[4] Wulf, W. A., Cohen, E., Corwin, W., Jones, A., Levin, R., Pierson, C., Pollack, F., "Hydra: The Kernel of a Multiprocessor Operating System", CACM 17,6 (June 1974), pp. 337-345.

[5] Fuller, S. H., Almes, G. T., Broadley, W. H., Forgy, C. L., Karlton, P. L., Lesser, V. R., Teter, J. R., "PDP-11/40E Microprogramming Reference Manual," Technical Report, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pa., January 1976.

[6] Wulf, W. A., Levin, R. and Pierson, C., "Overview of the Hydra Operating System", proceedings of the 5th Symposium on Operating System Principles, Austin, Texas, Nov. 1975, pp. 122-131.

[7] Wulf, W. A. and Harbison, S. P., "Reflections in a Pool of Processors, An Experience Report on C.mmp/Hydra," Technical Report, Department of Computer Science, Carnegie-Mellon University, Pittsburgh, Pa., January, 1978.

[8] Pollack, F. J., A Design Methodology for Fault-Tolerant Software, Ph. D. thesis, Carnegie-Mellon University, 1978.

[9] Parnas, D. L., Response to Detected Errors in Well-Structured Programs, Carnegie-Mellon University Computer Science Department Report, 1972.

[10] R. J. Swan, "Switching Structure and Addressing Architecture of an Extensible Mutiprocessor: Cm*", Carnegie-Mellon University Computer Science Dept. Tech. Report, May 1978.

[11] S. H. Fuller, A. K. Jones, and I. Durham, (Eds.), "The Cm* review report", Carnegie-Mellon University Computer Science Dept. Tech. Report, June 1977.

[12] R. J. Swan, S. H. Fuller, D. P. Siewiorek, "Cm*: a Modular, Multi-Microprocessor", AFIPS Conf. Proc., Vol 46, pp. 637-644, 1977.

[13] R. J. Swan, A. Bechtolsheim, K. Lai, and J. Ousterhout, "The implementation of the Cm* Multi-Microprocessor", AFIPS Conf. Proc., Vol 46, pp. 645-655, 1977.

[14] S. H. Fuller, J. Ousterhout, L. Raskin, P. Rubinfeld, P. Sindhu, and R. Swan, "Multi-microprocessors: an overview and working example", Proc. of IEEE, Vol. 66, No. 2, pp. 216-228, Feb. 1978.

[15] A. K. Jones, R. J. Chansler, I. Durham, P. Feiler, D. Scelza, K. Schwans, and S. Vegdahl, "Programming issues raised by a multi-microprocessor", Proc. of IEEE, Vol. 66, No. 2, pp. 229-237, Feb. 1978.

[16] A. K. Jones, R. J. Chansler, I. Durham, P. Feiler, and K. Schwans, "Software management of Cm*, a distributed multiprocessor", AFIPS Conf. Proc., Vol 46, pp. 657-663, 1977.

[17] D. P. Siewiorek, D. E. Thomas, and D. L. Scharfetter, "Trends and Limitations on the Use of LSI Modules in Computer Structures", to appear in Computer, July, 1978.

[18] C. G. Bell, D. P. Bhandarkar, D. L. Feucht, S. L. Rege, and D. P. Siewiorek, "Large Scale Intergration - A designer's Viewpoint", Carnegie-Mellon University Computer Science Dept. Tech. Report, Nov. 1972.

[18] A. Ingle, and D. P. Siewiorek, "Reliability Modeling of Multiprocessor Structures", Proc. IEEE CompCon '76, Sep. 1976.

[19] C. G. Bell, R. C. Chen, S. H. Fuller, J. Grason, S. Rege, and D. P. Siewiorek, "The Architecture and Application of Computer Modules: a Set of Components for Digital System Design", Seventh Annual IEEE Comp. Soc. Int. Conf., Compcon '73, pp. 177-180, Feb. 1973.

[20] S. H. Fuller, D. P. Siewiorek, and R. J. Swan, "Computer Modules: An Architecture for Large Digital Modules", Proc. of the First Annual Symposium on Computer Architecture, pp. 237-241, Dec. 1973.

[21] S. H. Fuller, D. P. Siewiorek, and R. Swan, "Computer Modules - An Architecture for a Modular Multi-Microprocessor", Proc. of the ACM Annual Conf., pp. 129-133, Oct. 1975.

[22] H. Bellis, "Comparing Analytical reliability Models to Hard and Transient Failure Data", M. S. E. E. Thesis, Department of Electrical Engineering, Carnegie-Mellon University, April 1978.

[23] D. Scelza, "Cm* Host User Manual", Carnegie-Mellon University, Computer Science Dept., July, 1977

[24] Digital Equipment Corporation, "DZKMA - MOS/Core Memory Exerciser for PDP-11", Product Code: MAINDEC-11-DZKMA-A-D, Nov. 1975.

[25] Digital Equipment Corporation, "DVKAA - LSI-11 Basic Instruction Tests", Product Code: MAINDEC-11-DVKAA-A-D, Aug. 1975.

[26] Digital Equipment Corporation, "DVKAD - LSI-11 Test of Interrupts, Traps, Reset, and Wait Instructions"," Product Code: MAINDEC-11-DVKAD-A, Aug. 1975.

[28] Siewiorek, D.P., M. Canepa, and S. Clark, "C.vmp: The Analysis, Architecture, and Implementation of a Fault Tolerant Multiprocessor", Departments of Electrical Engineering and Computer Science, Carnegie-Mellon University, December 1976.

[29] Hopkins, A.L., Jr., and T.B. Smith, "The Architectural Elements of a Symmetric Fault Tolerant Multiprocessor", IEEE Transactions on Computers, v.C-24 no.5, May 1975, pp.498-505.

[30] Wakerly, J.F., "Microcomputer Reliability Improvement Using Triple-Modular Redundancy", Proceedings of the IEEE, v.64 no.6, June 1976, pp.889-895.

[31] Lunde, A., "Evaluation of Instruction Set Processor Architecture by Program Tracing", Communications of the ACM, February 1977.

[32] "LSI-11 PDP-11/03 User's Manual", Digital Equipment Corporation, 1975.

[33] "RXV-11 User's Manual", Digital Equipment Corporation, 1975.

[34] Chaney, T.J., S.M. Ornstein, and W.M. Littlefield, "Beware the Synchronizer", Compcon 1972, pp.317,319.

[35] Snow, E., and D.P. Siewiorek, "Impact of Implementation Design Tradeoffs on Performance: The PDP-11, A Case Study", Departments of Electrical Engineering and Computer Science, Carnegie-Mellon University, July 1977.

A Case Study of C.mmp, Cm*, and C.vmp -- II. Predicting and Calibrating Reliability of Multiprocessor Systems.

Daniel P. Siewiorek, Vittal Kini,

Rostam Joobbani, Harold Bellis.

Departments of Computer Science
and
Electrical Engineering
Carnegie-Mellon University
Pittsburgh, Pa.

Abstract

This paper focuses on measurement and modelling of hard failures in multiprocessors. The failure rate predictions of the Military Standardization Handbook 217B (MIL 217B) are compared with semiconductor chip vendor data and data from Carnegie-Mellon University's multiprocessor systems. Based on these comparisons a modified MIL 217B model is proposed. The modified model is employed to calculate module failure rates for the three multiprocessors designed, implemented and currently operating at C-MU. Hard failure reliability models for these three systems are presented. These models use the calculated module failure rates as a basis for a consistent comparison of the three systems.

Keywords

Multiprocessor, Hard failure, Reliability Models, Failure Rate Prediction

This work was supported in part by the Advanced Research Projects Agency under contract number F44620-73-C-0074, which is monitored by the Air Force Office of Scientific Research; in part by the Office of Naval Research under contract number NR-048-645; in part by the National Science Foundation Grant GJ 32758X; and in part by the Defence Communications Agency under contract number DCA100-76-C-0058.

Table of Contents

1. Introduction

A companion paper [1] has presented the architecture and reliability experience of three multiprocessor systems constructed at Carnegie-Mellon University. Data on transient failures in the various systems was also presented.

This paper focuses on measurement and modeling of hard failures. Section 2 deals with the modeling of non-redundant systems. First the Military Standardization Handbook 217B (MIL 217B) is presented as a way of estimating hard failure rates. The model predictions are compared to semiconductor chip vendor data and data from C-MU's multiprocessor systems. Based on these comparisons, a modified MIL 217B model is proposed and used in the remainder of the paper. A program called AUTOFAIL allows the parameterized modificaton of MIL 217B using engineering drawing information as input. AUTOFAIL was used to produce the complexity and failure rate tables that appear in the various modeling sections.

Section 3 discusses levels of reliability modeling and the assumptions used in developing the system failure on exhaustion models. The final three sections contain the details of the reliability model for each architecture. The sections all follow the same format: brief description of the architecture, effect of various component failures, component failure rates, reliability models, reliability curves, and discussion. The uniform treatment of the architectures allows a more consistent comparison of the results.

2. A Hard Failure Reliability Model and Its Calibration

In order to compare the various multiprocessor structures, a uniform hard failure reliability model is required. The model should also explain the actual observed failure rates. This section introduces the MIL model 217B and two separate approaches to its calibration: data obtained from accelerated life testing of chips and life failure data from Cm*. From this calibration, a modified MIL model 217B emerged and was used to model all three multiprocessor systems. A failure rate calculation program, AUTOFAIL, is also described. The automation of the failure rate calculation insures accuracy and allows experimentation with the model versus observed data.

2.1. MIL Model 217B

It is usually assumed that the failure of electronic components follow the Poisson distribution with failure rate λ. That is:

1. Probability of transition from state with n occurrences to n+1 occurrences in time Δt is:

$$\lambda \Delta t$$

2. Occurrences are independent;

3. Transition probability of two or more occurrences in the interval Δt is neglected.

Then

Probability of k failures in time [0,t] = $e^{-\lambda t}(\lambda t)^k/k!$

Reliability = probability of no failures in time [0,t] = $e^{-\lambda t}$

With these assumptions, if a system does not contain any redundancy (i.e., every component must function properly for the system to work), and if component failures are statistically independent, then the system reliability is also exponential. Furthermore, the failure rate of the system is the sum of the failure rates of the individual components. This is also referred to as the Parts Count Model.

The Reliability Analysis Center has extensively studied statistics with respect to electronic component failures. The data has led to the development of a widely used reliability model for chip failures. The following is a sketch of the model presented in the Military Standardization Handbook 217-B [2].

The failure rate model for a single chip takes the form:

$$\lambda = \pi_L \pi_Q (C_1 \pi_T + C_2 \pi_E)$$

where

π_L A learning factor based on the maturity of the process. It assumes values of 1 or 10.

π_Q A quality factor based on incoming screening of components. Values range from 1 to 150.

π_T A factor based on the ambient operating temperature and the type of semiconductor process. Values range from 0.1 to 1000.

π_E A factor based on the operating environment. Values range from 0.2 to 10.

C_1, C_2 Complexity factors based on the number of gates (random logic) or number of bits (memory) in the component.

Since the rate of technology advance is rapid, new component types are continually being introduced. In addition the "learning curve" for any particular component type changes with experience engendered by its use in the field. There is then some question as to the accuracy of MIL Hdbk 217B, particularly with regard to newer technologies such as MOS RAMs, and ROMs.

Typical component failure rates are in the range 0.1 to 1.0 per million hours. Thus tens of millions of component-hours are required to gain statistically significant results. Two separate approaches were used to gather sufficient data for comparison with the MIL 217B model: life cycle testing of components and analyzing field repair information. The following subsections summarize the results.

2.2. Life Cycle Testing

In this approach a small number of components are tested in a controlled environment. Frequently an elevated temperature is used to accelerate failure mechanisms. A translation factor is then used to equate one hour at elevated temperature to a number of hours at ambient. The translation factor is usually derived from the Arrhenius Equation:

$$R = Ae^{-E_a/KT}$$

where

R = reaction rate constant

A = a constant

E_a = activation energy

K = Boltzmann's constant

T = Absolute temperature.

Often these accelerating factors are extrapolated into regions (such as ambient) where there is very little corroborating data. Because of the exponential, accelerating factors can become quite large.

In addition, there is little consensus on the appropriate activation energy. Activation energies of 0.23 eV to 1.92 eV have been used. The temperature dependent portion (π_T) of MIL Hdbk 217B assumes an activation energy of 0.41 eV while MIL Std 883A (used to qualify components for procurement) assumes 1.02 eV. Consider conversion from 125°C to 50°C. The ratio of the MIL Std 883A acceleration factor to the MIL Hdbk 217B is 61.93. This means a factor of 62 difference in predicted failure rate (λ) from the same life cycle test data!

Furthermore, only one activation energy is assumed. In reality many different mechanisms contribute to chip failure and they are not all accelerated by the same amount for the same temperature increment. Assuming a single activation energy can lead to substantial errors, especially when using extrapolation.

Returning to the form of the MIL Hdbk 217B model we see that high temperature testing only calibrates the temperature portion. The environmental portion (aging and mechanical stress), which can range from 10% (high temperature) to 70% (low temperature) of the predicted failure rate, is not measured. One last problem with using high temperature life cycle testing is that semiconductor manufacturers usually lump test data by process (i.e. bipolar, MOS, etc.) thus hindering a comparison with the MIL Hdbk 217B complexity factors.

Given the problems listed above, data from several sources was combined using assumptions to establish commonality. The data represents over 3 billion hours of real time operation (of which 137 million hours were high temperature testing). The data sources were:

RADC — A list of life cycle test data as a function of device complexity. Most were from high temperature testing and some data about test temperatures was missing.

Signetics — High temperature testing with data lumped by process. Some individual test data by component number but usually a small number of component-hours. An activation energy of 0.41 eV is assumed and calibrated by experiment for bipolar component temperature translation.

SandersAssociates - Analysis of field data.

Using a junction temperature of 50°C, a temperature accelerating factor

corresponding to 0.41 eV activation energy, and adding in the MIL Hdbk 217B predicted environmental portion, Figure 1 results. The RADC data is raw and was not temperature translated since a significant percentage of the data did not have a test temperature recorded. The two anomalous points in the RADC data at 20 and 58 gates should be treated as suspect since these two points had the least number of test hours (less than a million).

The temperature translated data in Figure 1 tracks the MIL Hdbk 217B model generally within a factor of two while the Sanders Associates data is in close agreement

Since RAM and ROM data is less extensive, it is reproduced in Table 1 along with a few points of MOS data. The Signetics data was temperature translated to 50°C. The total failure rate and temperature dependent portion are listed separately so that comparison with high temperature, translated test data is facilitated. The Signetics data with a less than (<) sign is an upper bound in cases where no failures were observed.

For bipolar RAMs and ROMs the MIL Hdbk 217B model for total failure rate tracks within a factor of two and is generally pessimistic. The temperature portion roughly tracks but in a less precise manner. It should be noted that the majority of this data is from one source (i.e. Signetics).

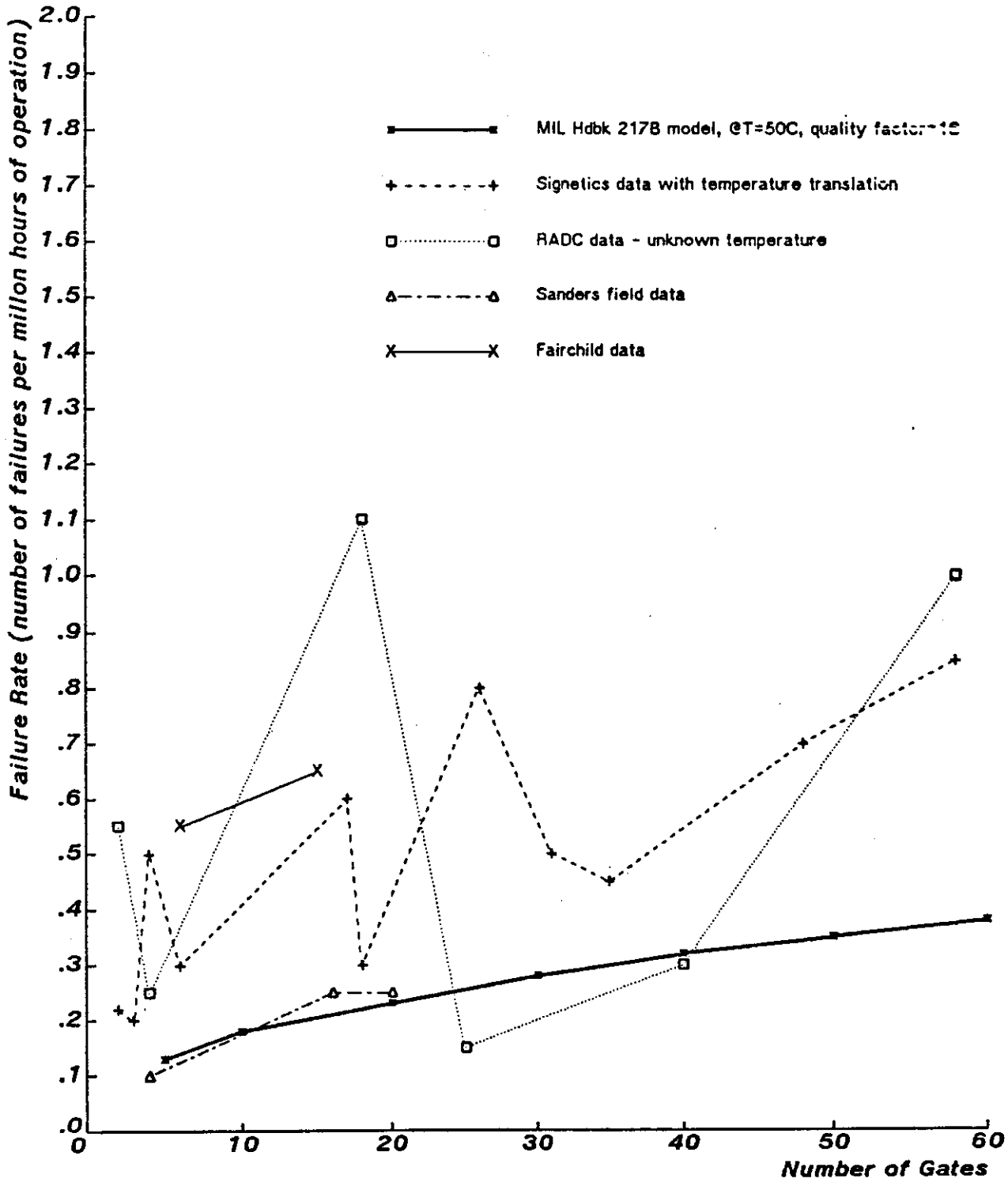


Figure 1. Data from Life Cycle Testing

Table 1. ROM, RAM, and LSI Life Cycle Test Data.

Part Description	Source	Failure Rate Observed per Million Hours		Failure Rate from Mil Std 217B per Million Hours		Failure Rate from Mil Std 217B per Million Hours Reduced by a Factor of 16 in Bits		Failure Rate from Mil Std 217B per Million Hours Reduced by a Factor of 64 in Bits
		Temperature Portion	Total	Temperature Portion	Total	Temperature Portion	Total	Total
Bipolar RAMs								
256 bits	Sanders Associates	-----	1.28	-----	.635	-----	.113	-----
256 bits	*Signetics	.078	.398	.313	.635	.059	.113	-----
576 bits	*Signetics	<.544	<.797	.511	1.000	.096	.173	-----
1K bits	*Signetics	.068	.852	.723	1.51	.267	.136	-----
Bipolar ROMs								
256 bits	*Signetics	<.44	<.668	.179	.363	.034	.064	-----
1K bits	*Signetics	.211	.659	.414	.865	.078	.153	-----
2K bits	*Signetics	1.75	2.45	.629	1.33	.118	.236	-----
4K bits	*Signetics	.053	1.173	.955	2.06	.179	.364	-----
Schottky PROMs								
256 bits	+*RAC	.073	.265	.179	.363	.034	.064	-----
1K bits	+*RAC	1.14	1.588	.414	.865	.078	.153	-----
MOS RAMs								
1K bits	Sanders Associates	-----	.194	-----	2.504	-----	.454	.193
MOS ROMs								
1K bits	Sanders Associates	-----	.078	-----	1.433	-----	.26	.111
MOS Random Logic								
8080 Microprocessor	+*RAC	-----	.418	-----	-----	-----	.616	.293

* Temperature translation to 50 °C

+ Reliability Analysis Center, RADC

For MOS RAMs, ROMs, and random logic there is even less data but it clearly indicates the MIL Hdbk 217B model is a factor of 16-64 pessimistic. Since the 217B model was published in 1974 and was probably developed on 1972 data, one might speculate that MOS technology might not have settled down in time for creation of the model. There are many parameters that can be altered in 217B to take into account process maturity. One could modify a constant factor such as π_Q (i.e. move the curves up and down). One could speculate that the complexity factor should be modified with time since as the process matures more complex components are feasible (i.e. move the curves to the right). If we use the rule of thumb that memory doubles in complexity every 1-1.5 years and we want the state-of-the-art portion of the curve to correspond in 1977 to where it was in 1972, then the complexity axis (number of bits) should be divided by $2^4 = 16$. This modified 217B model is shown in the last column of Table 1. The modified 217B does rather poorly on bipolar components but is within a factor of 3 on MOS components.

2.3. Analysis of Hard Failure Data

In this approach information about total systems is analyzed and broken down into failure rate by components. The major difficulties are lack of control over the environments of the systems and incomplete data.

The various systems will be of different configuration, and subjected to different environments (π_E), operating temperatures (π_T), and power-on time (affecting the calculated failure rate). In addition, current repair practices do not lend themselves to component level data analysis. Typically a repairman will fix a system by board swapping. The boards are then sent to a repair depot where they lose their identities, and where repair actions are often not recorded.

Furthermore, the repair activity may induce additional or future failures. However, with careful planning and documentation these difficulties can be overcome. In our case we carefully collected hard failure data from the Cm* error logs [3]. The data presented in the following tables and figures were collected through May 10, 1978.

The Mean Time Between Failures (MTBF) was calculated assuming failures were independent. The MTBF was obtained by dividing the total time by the total errors. Because of the small number of failures per module, a concept called "module time" was introduced. Module time allows data from all modules to be combined. If there are k modules running during a period of time then

$$\text{module time} = \sum_{1 \le i \le k} t_i$$

where t_i is the time the i^{th} module was up. Now assuming that all the modules of a type are identical, then the times that failures were recorded in real time can be transfered to a "typical" module in module time. Table 2 depicts this module time data for Cm*.

The complexity in chips referenced in Table 2 is a measure of the actual utilization of chips per module. In the case of the LSI-11 (DEC), the actual number of chip sockets used is 76, of these 72 contain digital integerated circuits. The number of chips used is recorded as 68, which implies that the unused functions on the chips add up to 4 chips.

Module	Complexity (Chips)	# of Modules	Total time (Hours)	Total failures	MTBF (Hours)
Kbus	138	3	36696	8	4587
Pmap	106	3	37416	12	3118
Mmicro	116	6	68328	4	17082
Mdata	142	3	37080	2	18540
Linc	116	3	22608	0	-
LSI-11	68	14	163200	10	16320
Slocal	126	10	120720	5	24144
4K memory	56	21	260568	5	52003.6
16K memory	104	10	122280	5	24456
Slu	28	17	223248	5	44649.6
Power board	6	16	195456	3	65152
Refresh	14	16	162912	0	-

Table 2. Failure Data on Cm*

An analysis of the variance of the error log data showed that uncertainties associated with module commissioning dates (i.e. initial power up and integration into the system) were insignificant.

The next step was to determine the failure distribution from the data. There are two basic approaches. The first is to determine the instantanous failure rate or hazard function, which indicates the failure distribution. The second method is to use statistical tests to differentiate between distributions.

The following equation is used for plotting a piecewise linear graph of the hazard rate:

$$\text{Hazard rate} \quad z(t) = \frac{(n(t) - n(t + \Delta t))/n(t)}{\Delta t}$$

The number of survivors at any time is given by n(t). The choice of Δt is not specified and is occasionally chosen to end just after each failure. Another method of choosing the size of Δt that smooths out the curve is to divide the total time into equally spaced intervals. The number of intervals is given by the following equation [4]:

$$k=1 + 3.3\log_{10}M$$

where k is the number of intervals and M is the number of failures. This latter method was used for the plotting of data on the modules.

Data for these hazard calculations is commonly obtained through life tests. The data that was obtained from Cm* differed from that of a life test in that when a failure was detected in a module, the module was repaired and placed back in operation. Thus some components in the module were starting out their operational life while others were in intermediate stages. A second difference is that modules had different amounts of operating time. Due to the few failures detected and the small number of modules being tested, all the failure data must be used. To accommodate the data on Cm*, a replacement assumption is necessary.

The replacement assumption posits that a repaired module can be considered to be new. The concept of module time described earlier is then used along with this assumption to make effective use of the small amount of available data. For example, consider the case of some set of modules $\{M_i\}$. Each time some M_i fails, it is repaired and is considered to be new using the replacement assumption. The jth such "incarnation" of M_i can be considered to be a new "virtual" module $M_{i,j}$ which has a lifetime of t_{ij} before it fails and is later reincarnated as the new virtual module $M_{i,j+1}$. Then at any given time the set of virtual modules $\{M_{i,j}\}$ is such that each member of the set has either suffered one incapacitating failure or has not failed at all. Module time for this set is then given by

$$t_m = \sum_{i,j} t_{ij}$$

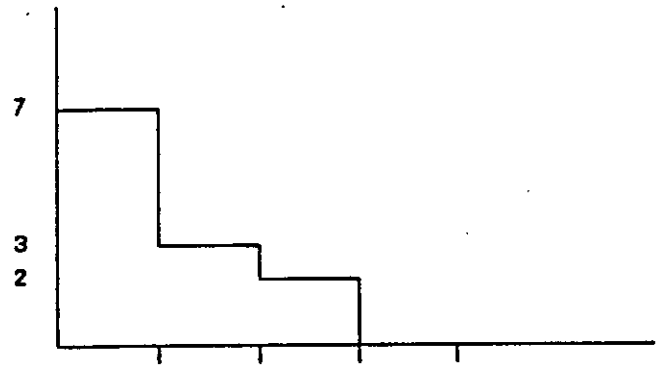
A "typical" module of the set $\{M_i\}$ is then assumed to have been in use for time t_m and have suffered the same number of failures as the set $\{M_i\}$ taken as a whole. The hazard rate expression previously mentioned is then redefined as follows:

$$\text{Hazard rate } z(t) = \frac{F(t,t+\Delta t)/n(t)}{\Delta t}$$

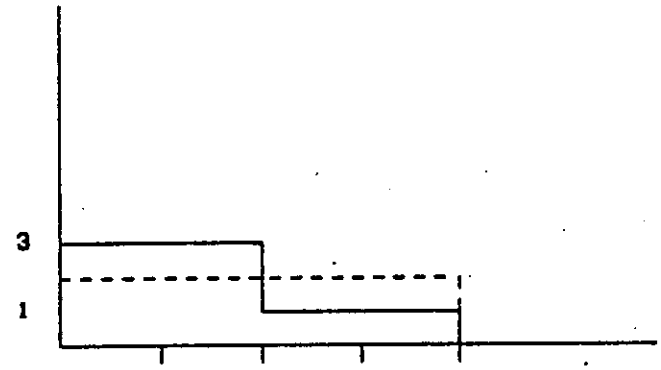
where $F(t,t+\Delta t)$ is the number of failures between time t and time $t+\Delta t$. For these cases $n(t)$ is always equal to one i.e. the "typical" module.

There was only enough data on the modules to construct four rough hazard functions. They are of the Pmap, the Kbus, the LSI-11, and of the total Cm* system (Figures 2 and 3).

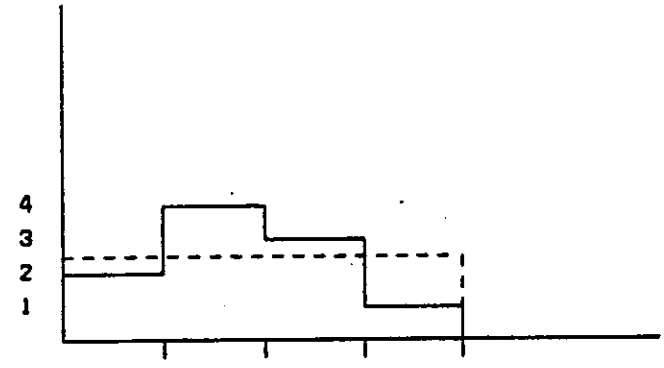
HAZARD FUNCTIONS



Pmap Interval = 309.8 Days



Kbus Interval = 382.25 Days



LSI-11 Interval = 1700 Days

Figure 2. Hazard Curves for Pmap, Kbus and LSI-11.

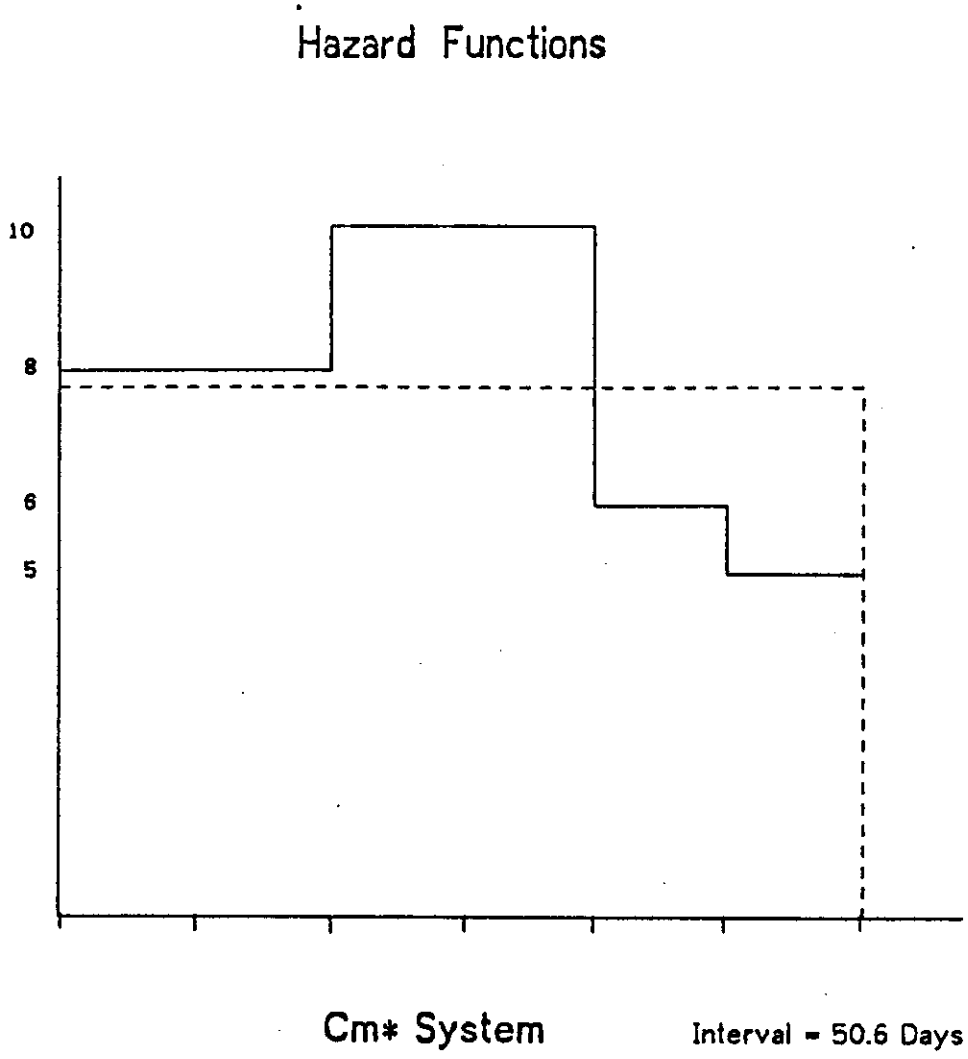


Figure 3. Hazard Curve for total Cm* system.

The graph of the Pmap appears to exhibit a decreasing hazard rate. This indicates a problem with infant mortality. A note should be made here that nine out of the twelve failures on the Pmap were attributed to one chip type, the 74373.

The Kbus seems to display either a decreasing or constant hazard rate. Assuming it to be constant, its value would be around two failures per 382.25 day interval, which corresponds to a MTTF of 191.125 days.

The LSI-11 was the other module examined. This curve indicates a possible constant hazard rate 2.5 failures per 1700 day interval or a MTTF of 680 days.

The final hazard function is presented in Figure 3 is that of the system using all the modules. It is plotted using the first 304 days since commissioning for all the modules. Over this period, a MTBF of 155.2 hours is indicated.

The MTTF presented in Table 2 were calculated by dividing the total time by the total number of failures. In the case of a constant hazard rate, the MTTF was calculated by dividing the length of an interval by the average number of failures per interval. That these two calculations are equivalent can be seen from:

MTTF for constant hazard rate =

(length of interval)/(average failures per interval)=

(length of interval)/((total failures)/(number of intervals))=

(total time)/(total failures)=

MTTF from Table 2

The results presented have been inconclusive in predicting the failure distribution. The exponential distribution is plausible, but a better test for the data is needed. To accomplish this the data is fit to a generalized distribution that has the Exponential as a special case. A generalized distribution that is used in reliability studies is the Weibull for which the probability density function is given by:

$$f(x) = \frac{\beta}{\eta}\left(\frac{x}{\eta}\right)^{\beta-1}\exp\left(\frac{x}{\eta}\right)^{\beta}$$

This degenerates to the exponential distribution when $\beta=1$. Table 3, presents the maximum likelihood estimates for β and η and the 95% and 68% confidence interval on β for the different modules.

The 95% (68%) confidence interval means that if the experiment were repeated 100 times, on the average 95 (68) times β would lie in the given range. In order to tighten up the range on β, a smaller confidence interval is used. The data in Table 3 indicates a wide spread in the maximum likelihood estimates of β, but in all but two cases $\beta=1$ is enclosed in the 95% confidence interval. The 68% confidence interval is only able to enclose a $\beta=1$ for half of the modules. This means that while an exponential failure distribution is plausible, actual data presents enough variation that the impact on the system of an exponential failure assumption should be examined. It should be emphasised that the above parameters were estimated using a small number of data points. The numbers will be refined as more data becomes available.

Module	η	β	95% Confidence Interval on β ($\beta \pm 1.96\sqrt{V(\beta)}$)	68% Confidence Interval on β ($\beta \pm \sqrt{V(\beta)}$)
Kbus	189.5	.721	.30 : 1.15	.50 : .94
Pmap	104.0	.537	.29 : .79	.41 : .66
Mmicro	625.9	1.264	.23 : 2.30	.73 : 1.79
Mdata	3043.9	.344	0.0 : .79	.12 : .57
LSI-11	716.1	.915	.41 : 1.42	.66 : 1.17
Slocal	1977.1	.584	.1 : 1.07	.34 : .83
4 K memory	1496.4	1.320	.28 : 2.36	.79 : 1.85
16 K memory	690.7	1.945	.40 : 3.50	1.15 : 2.74
Slu	1320.6	1.348	.25 : 3.08	.79 : 1.91
Power board	1819.4	1.295	0.0 : 2.67	.59 : 2.00

Table 3. Estimated Parameters of the Weibull from Failure Data

The exponential distribution was chosen to model the different types of modules in the Cm* system. This decision can be supported by the observations of Figures 2 and 3. The next step is to determine the parameters of the chosen distribution.

Table 4 gives the maximum-likelihood estimator (MLE) of λ and its 50% confidence interval. Again, it should be emphasized that this analysis has been based on a small number of failures. For conclusive results much more data is necessary.

Module	λ (Fail/10^6Hr)	MTTF (Hours)	50% Confidence Interval (on MTTF)
Kbus	218	4587	3397.8 : 6167.4
Pmap	320.7	3118	2461.6 : 3938.5
Mmicro	58.5	17082	10932.5 : 26953.9
Mdata	53.9	18540	9459.2 : 38625.0
Linc	-	-	-
LSI-11	61.3	16320	12553.9 : 21058.1
Slocal	41.4	24144	16313.5 : 35822.0
4K memory	19.2	52113.6	35211.9 : 77319.9
16K memory	40.9	24456	16524.3 : 36284.9
Slu	22.4	44649.6	30168.7 : 66245.7
Power board	15.3	65152	38324.7 : 113307.8
Refresh	-	-	-

Table 4. Calculated Failure Rates from Data on Cm*

Module	Complexity (Chips)	Quality factor 16/16	16	150	150/16
Kbus	138	44.1	53.3	499.3	413
Pmap	106	35.6	39.6	371.7	333.7
Mmicro	116	26.6	128.3	1203	249.2
Mdata	142	35.4	146.5	1373.8	332.4
Linc	116	35.5	75.1	704.6	332.8
LSI-11	68	29.9	379350.8	35568289.0	280.3
Slocal	126	27.4	31.8	298.4	256.8
4K memory	56	23.1	99.8	936	216.9
16K memory	104	74.1	380.9	3571.1	694.7
Slu	28	4.7	8.7	81.6	43.9
Power board	6	.97	.97	9.1	9.1
Refresh	14	2.6	2.6	24.9	24.9

Table 5. Predicted Failure Rates for Cm* Components

Module	Failure Rate	Best Fit	Predicted Failure Rate
Kbus	218	Q = 150/16	413
Pmap	320.7	Q = 150/16	333.7
Mmicro	58.5	Q = 16/16	26.6
Mdata	53.9	Q = 16/16	35.4
Linc	-		
LSI-11	61.3	Q = 16/16	29.9
Slocal	41.4	Q = 16	31.8
4K memory	19.2	Q = 16/16	23.1
16K memory	40.9	Q = 16/16	74.1
Slu	22.4	Q = 150/16	43.9
Power board	15.3	Q = 150/16	9.1
Refresh	-		

Table 6. Results of Maximum Likelihood Ratio Test

Four variants of the Mil 217B model were selected for comparison: quality factors of 16 and 150; LSI chip complexity derating of 1 and 16. The predicted failure rates are shown in Table 5. The results of the comparison of the data to various parameter changes is shown in Table 6. They consist of the observed failure rate , the best fitting variant of the Parts Count Model examined, and its associated failure rate prediction. This table indicates that the modules tend towards a derating of the quality factor by 16 for MOS chips. This coincides with the conclusion from life cycle test data mentioned earlier.

The data on the Pmap indicates a quality factor of 150 with a derating factor of 16. As was noted earlier, 9 of the 12 failures were attributed to a single chip type, the 74373. There are seven 74373 chips in each of the three Pmaps. The MIL 217B model predicts that 6.7% of the failures for the Pmap will be due to this chip. The failure rate observed for the 74373's in the Pmap was nine failures in 37416 hours or 240.5

failures per million hours. This corresponds to a quality factor for the 74373's of 516, which suggests a possible bad batch of chips. Using only the other failures to calculate a failure rate produces a λ = 80.2 failures/10^6 hrs. This corresponds to a quality factor of 36, which lies between 150 and 16.

The Slocal is best fit by a quality factor of 16. If a derating of 16 is assumed then the quality factor for the Slocal lies between 150 and 16. In fact all but the memory boards that have a quality factor which is just less than 16, and the power boards, which are slightly above 150, lie within the range of 150 to 16. In general, industrially produced components (in this case by Digital Equipment Corp.) indicate a quality factor close to 16. CMU-built components exhibit a quality factor of 16 for more mass produced components (Mmicro and Mdata are printed circuit board RAMs also used for writable control stores on C.mmp; the Slocal has been through two design cycles) and a quality factor of 150 for the remainder.

The expected failure rate for a system composed of all the modules using their appropriate quality factors from Table 6 is 7222.2 failures per 10^6 hours. This is equivalent to a MTTF of 138.46 hours, which may be compared to the MTTF of 155.2 Hours derived from the hazard curve in Figure 3.

2.4. AUTOFAIL--Automated Failure Rate Calculation

A program, AUTOFAIL, has been written at CMU [5] that simplifies the procedure of computing a system's failure rate from the failure rates of its constituent parts as predicted by the MIL model 217B. A system may be described to AUTOFAIL in the form of a list of chips and/or subsystems, which are likewise recursively defined. Figure 4 is the input description of the DEC LSI-11 microcomputer. Parameters such as the various π factors may be modified so as to obtain a sensitivity analysis. The format of this file is:

```
[ Module Name
  Body]
```

where "Body" is a listing of all chips and submodules making up this module. A chip is identified by an integer, specifying the number of chips of this type used, or by an integer followed by an "F", specifying the number of functions of this chip type that were used. This is then followed by a comma and the name of the chip. Submodules are constructed using the same format as modules.

```
[LSI11
  [SPECIAL.FUNTIONS
    2F,DM8641
    3F,7474
    1 ,7442
    5F,7404
    1F,7400]
  [BUS.ARBITRATION.LOGIC
    1F,7400
    1F,DM8837
    3F,7474
    1F,DM8641]
  [INTERRUPT.CONTROL.AND.RESET.LOGIC
    4F,7404
    4F,7474
    2F,DM8641
    2F,7400
    5F,DM8837
    1F,7405
    1F,74174]
  [CLOCK.PULSE.GENERATOR
    1F,7400
    1F,74140
    2F,7474
    1F,74139
    6F,7404
    4F,MH0026]
  [ROM.CHIPS
    3 ,CP1631B]
  [DATA.CHIP
    1 ,CP1611B]
  [CONTROL.CHIP
    1 ,CP1621B]
  [BUS.DRIVERS.AND.RECIEVERS
    4 ,74257
    4 ,DM8641
    1F,DM8641
    4F,7411
    2F,7405]
  [MEMORY
    16,MK4096]
  [BUS.I/O.CONTROL.LOGIC
    1F,7497
    7F,7400
    7F,7404
    2F,7411
    4F,7474
    5F,7410
    5F,DM8641
    1F,DM8837]
```

(Continued)

Figure 4. LSI-11 Input File for AUTOFAIL

```
[I/O.BUS.MEM.READ.DATA.MUX
  4F,7475
  2F,74257
  3F,7410
  3F,7400
  2F,74140
  2F,7405
  2F,741071
[FAST.DIN.MUX
  1F,74257
  1F,7400
  1F,74041]
```

Figure 4. LSI-11 Input File for AUTOFAIL (Continued)

Figure 5 is a listing of the output for the LSI-11 produced by AUTOFAIL. The top line consists of the values of the various derating factors used. The π values are presented on the following line. The failure rates for the LSI-11 and the submodules are shown along with the percentage of the failure rate for a module that is attributed to each submodule. In the case of a partially used chip (i.e. denoted by the number of functions "F"), AUTOFAIL prorates the chip failure rate by the fraction of the total number of functions used. It is sometimes desirable to examine the behavior of a particular chip or chip type. The lower table indicates this ability by listing the number of chips, failure rates, and percentages for the different chip types.

```
lsi11.rel[x330ds73]   LSI=    16.000   ROM=    16.000   RAM=    16.000
E =     1.000 Q =    16.000 L =     1.000 T =    25.000
```

MODULE	FAILURE RATE	PERCENTAGE
LSI11	29.893	100.000
SPECIAL.FUNTIONS	.669	2.237
BUS.ARBITRATION.LOGIC	.358	1.172
INTERRUPT.CONTROL.AND.RESET.LOGIC	.776	2.596
CLOCK.PULSE.GENERATOR	.851	2.847
ROM.CHIPS	3.413	11.416
DATA.CHIP	1.160	3.880
CONTROL.CHIP	1.160	3.880
BUS.DRIVERS.AND.RECIEVERS	1.588	5.314
MEMORY	16.991	56.837
BUS.I/O.CONTROL.LOGIC	1.500	5.019
I/O.BUS.MEM.READ.DATA.MUX	1.195	3.999
FAST.DIN.MUX	.241	.805

```
# of chips =     68.917  # of gates =   7145.883  # of bits =  99328.000
```

TYPE	# of CHIPS	FAILURE RATES	PERCENTAGE
SSI	37.250	4.899	16.387
MSI	18.667	2.272	7.600
LSI	2.000	2.320	7.760
ROM	3.000	3.413	11.416
RAM	16.000	16.991	56.837
MOS	21.000	22.723	76.013
BIP	47.917	7.171	23.987

```
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Figure 5. Output from AUTOFAIL for LSI-11

The parameters of the MIL model 217B can be varied by subsystem or even chip type so that variations in ambient temperature (i.e. a board near the power supply) or technology (i.e. a new chip for which parameters such as junction temperature may not be known) can be modeled. At the chip level it is also possible to modify the number of devices on a chip to gauge the effect of the size of a new chip type on the design. Further, individual chip type or entire chip class (i.e. ROM, RAM, MOS, LSI) can be assigned any arbitrary complexity derating factors. Again, this is used to test the sensitivity of the system failure rate as a function of the unknown parameter.

This program, AUTOFAIL, was used to generate the failure rates for all the multiprocessor components described in this paper. Actual parts lists were used as the input, and a uniform list of parameters (π_Q=16, π_E=1, π_L=1, ambient temperature = 25°C, division of all ROM, RAM, and LSI complexities by 16) was maintained throughout.

3. Techniques for introducing protective redundancy

There are three ways to introduce protective redundancy in to computer systems [6]:

Hardware- additional modules

Software- special programs

Time - replication of operations

Only hardware redundancy will be considered in this paper. Hardware redundancy schemes can be divided into two types:

Static redundancy- all the modules are powered up and functioning. The hardware failures are masked by the redundant modules.

Dynamic redundancy- There are two types of modules, active ones that directly contribute to the operation of the system, and standby spares.

In dynamic redundancy fault tolerance is achieved by three sequential actions: fault detection, diagnosis, and recovery. As a rule, detection or identification of faulty units is not perfect. Thus the probability of system survival is modeled by the probability of module failure and the conditional probability that the system recovers (e.g. correctly detects, isolates, and recovers) given a failure. This latter conditional probability has been termed coverage [7],[8]. It has been shown that coverage has a significant impact on the survival probability of a system.

In this paper we will model the C.mmp and Cm* multiprocessor systems developed at Carnegie-Mellon University as dynamic redundancy systems assuming perfect coverage (C.vmp employs static redundancy). We are primarily interested in predicting the maximum reliability achievable by the architectures (hence perfect coverage) with no arbitrary policy decisions (i.e. effort devoted to software diagnostic development, quality of programing staff, etc.). However, as more data becomes available on actual system failures, the models will be modified and calibrated. In particular, transient failures are at least an order of magnitude more frequent than permanent failures [1] and the models will be augmented to include transient behavior.

3.1 Levels in reliability modeling

Typically, a reliability model divides a structure into various subsections that are easier to study than the whole structure itself. There are certain levels at which it is customary to model systems [9].

1. The highest level of modeling is the system level. In this level the entire system is considered as a black box. Statistics are gathered about events (e.g. failure of a certain kind). A model then can be suggested to fit the data as closely as possible. An enormous amount of data is required for successful modeling.

2. The next level is the module level. The system is divided into a number of modules which have mutually independent failure probability distributions. The system model is obtained by a composition of the models for the modules.

3. The third level is the gate level. Gate reliability is often the basic parameter used to obtain the system reliability.

One rarely has to go below the gate level. However, if the redundancy is introduced at a lower level, the component level of modeling is required, where components are transistors, diodes, resistors, etc.

In evaluating the reliability of the three architectures, module level modeling is used. There are several reason for this choice. A first-order approximation of the reliability of a large hardware system can be easily derived by assuming module independence and counting components in each module. Also, the number of parameters are usually few and the dominating parameters or architectural features are easily identified. Furthermore, all of the redundancy in these three systems is at the module level thus allowing the various architectures to be compared. Usually this level of modeling is called PMS level (processor, memory, switch).

3.2 Redundancy model

Redundant systems can be modeled in one of several ways:

1. Redundancy with periodic maintenance. This type of modeling includes the effect of periodic maintenance on the reliability of the system.

2. Redundancy with repair. This introduces a second random variable, the time to do module repair, into the model, thereby complicating the analysis. However, under this model, the probability that the minimum required set of modules is functional at any given time is significantly higher than the first model. Consequently the reliability of such systems is much improved though there may be periods of degraded perfomance while failed modules are being repaired.

3. Redundancy with failure to exhaustion. This pessimistic and simplistic model assumes all redundant modules fail before any repair. The system is considered to be failed if it does not satsify the minimal set of functioning modules that comprised the corresponding minimal system. Repair is only done when a module failure causes system failure.

This paper uses the last method of redundancy modeling. Again, we are interested in predicting the maximum reliability achievable with no arbitrary policy decisions (i.e. mean time between maintenance periods, quality of the repair staff, etc).

As an example of failure to exhaustion, consider a system with five processors, ten memory modules and a clock for synchronization. Also consider a task running on this system that requires three processors and nine memory modules in order to run to completion. A minimal system with respect to the task requirements would contain exactly three processors and nine memory modules and the clock. This set of modules is the minimal module set for that particular task. The system being considered is therefore redundant with respect to the task requirements and will be considered failed only when it can no longer provide the minimal set of functioning modules. In other words the redundant system fails if the clock fails, or more than two processors fail, or more than one memory module fails.

Failure to exhaustion models typically enumerate all the states of the system (where a state is a pattern of failed and working modules) that meet or exceed the requirements of the minimal module set. If there are N identical modules with the reliability of each module R_c($R_c = e^{-\lambda t}$ where λ = failure rate) , and if a task requires k modules, the subsystem can tolerate

up to N-k failures, and the reliability of such a system is

$$R = \sum_{i=0}^{N-k} \binom{N}{i} R_o^{N-i} (1-R_o)^i$$

consider the reliability of the system mentioned in the previous example with the given task requirements.

Reliability of the system when all of the processors and memory modules are functioning is

$$R_0 = R_{clk} * R_{proc}^5 * R_{mem}^{10}$$

where

R_{clk} = clock reliability.

R_{proc} = processor reliability.

R_{mem} = memory module reliability.

Reliability of the system when one memory module fails is

$$R_1 = R_{clk} * R_{proc}^5 * \left[\binom{10}{1} R^9_{mem} (1 - R_{mem}) \right]$$

Reliability of the system when one processor fails is

$$R_2 = R_{clk} * \left[\binom{5}{1} R^4_{proc} (1-R_{proc}) \right] * R_{mem}^{10}$$

Reliability of the system when one processor and one memory fails is

$$R_3 = R_{clk} * \left[\binom{5}{1} R^4_{proc} (1-R_{proc}) \right] \left[\binom{10}{1} R^9_{mem} (1-R_{mem}) \right]$$

Reliability of the system when two processors fail is

$$R_4 = R_{clk} * \left[\binom{5}{2} R_{proc}^3 (1-R_{proc})^2 \right] R_{mem}^{10}$$

Reliability of the system when two processors and one memory fails is

$$R_5 = R_{clk} * \left[\binom{5}{2} R_{proc}^3 (1-R_{proc}) \right] \left[\binom{10}{1} R_{mem}^9 (1-R_{mem}) \right]$$

The total reliability is the sum of the reliabilities of the system in these six operational states:

$$R_{sys} = R_{clk} * \left[\sum_{i=0}^{2} \binom{5}{i} R_{proc}^{5-i} (1-R_{proc})^i \right] \left[\sum_{j=0}^{1} \binom{10}{j} R_{mem}^{10-j} (1-R_{mem})^j \right]$$

The following three subsections develop the hard failure models for the three multiprocessor systems. Due to its simplicity, the complete model is presented for C.vmp illustrating the development of the module reliabilities and total system equation. The model for C.mmp is more complex and its development is only outlined. The model for Cm* is very complex and only one of several possible functional system states is fully developed.

3.3 C.mmp, a Multi-miniprocessor

3.3.1 Architecture Summary

C.mmp is a canonical multiprocessor system with a 16X16 crosspoint switch (Figure 6). Up to 16 DEC PDP-11/40 processors may be connected to the processor ports on the switch. The 16 memory ports provide an address space in shared memory of 32 megabytes. Any processor can access any of the 16 memory ports thereby providing a maximum switch concurrency of 16 for memory accesses. The entire set of processors may communicate via an interprocessor bus which allows interprocessor interrupts at one of three priority levels, continuously broadcasts a 60-bit non-repeating clock value and allows any processor to halt, start or continue any other processor. Processor-generated 18-bit addresses are mapped onto a 25 bit physical address by the Dmap. The companion paper [1] in this issue provides a more detailed description of the C.mmp architecture.

3.3.2 Probabilistic Hard Failure Model For C.mmp

Figure 7 illustrates the PMS[1] model used for the reliability model. The effects of modules

[1] In PMS notation, P stands for processor; M for memory; S for switch; K for controller; A for arbiter; L for Link; and C for computer.

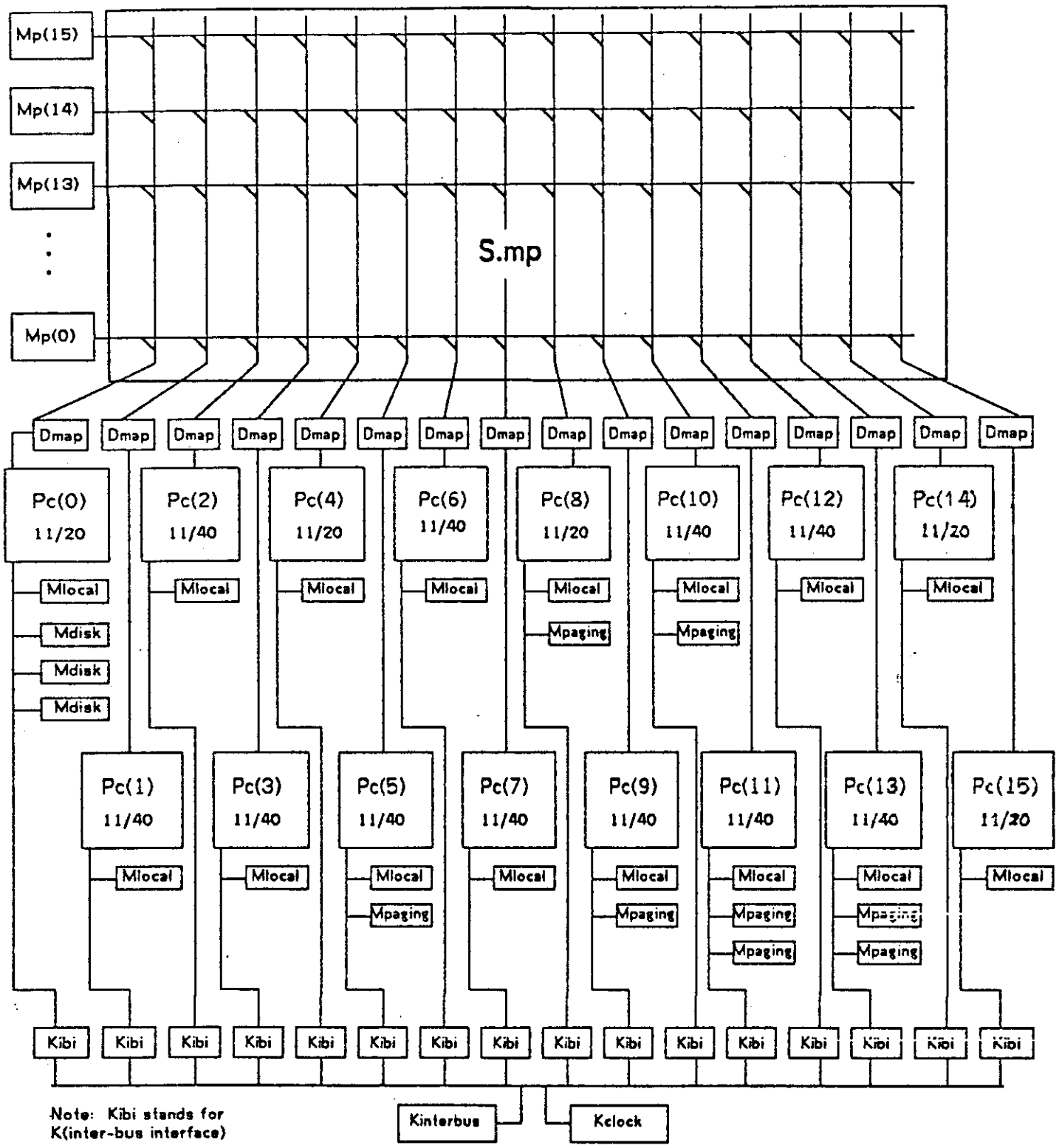


Figure 6. PMS Diagram for C.mmp

failures on the system are presented in Table 7. The architecture configuration parameters of the model are defined in Table 8. The minimum required modules to satisfy the operational requirements are parametrically. shown in Table 9. Table 10 lists the reliability parameters while Table 11 summarizes the failure rate of the various modules as calculated by AUTOFAIL in failures per million hours.

Module	Effect of failure
Processor	Loss of processor.
Processor controller	Loss of processor.
Local memory module	Loss of memory module.
Shared memory module	Loss of memory module.
Memory controller	Loss of memory port.
Memory arbiter	Loss of memory port.
Switch	Loss of the whole system.
Master clock	Loss of the whole system.

Table 7. Effect of module failures in C.mmp

P	Number of processors.
M	Number of shared memory modules/port.
N	Number of local memory modules/processor.
T	Number of memory ports.

Table 8. Architecture configuration parameters.

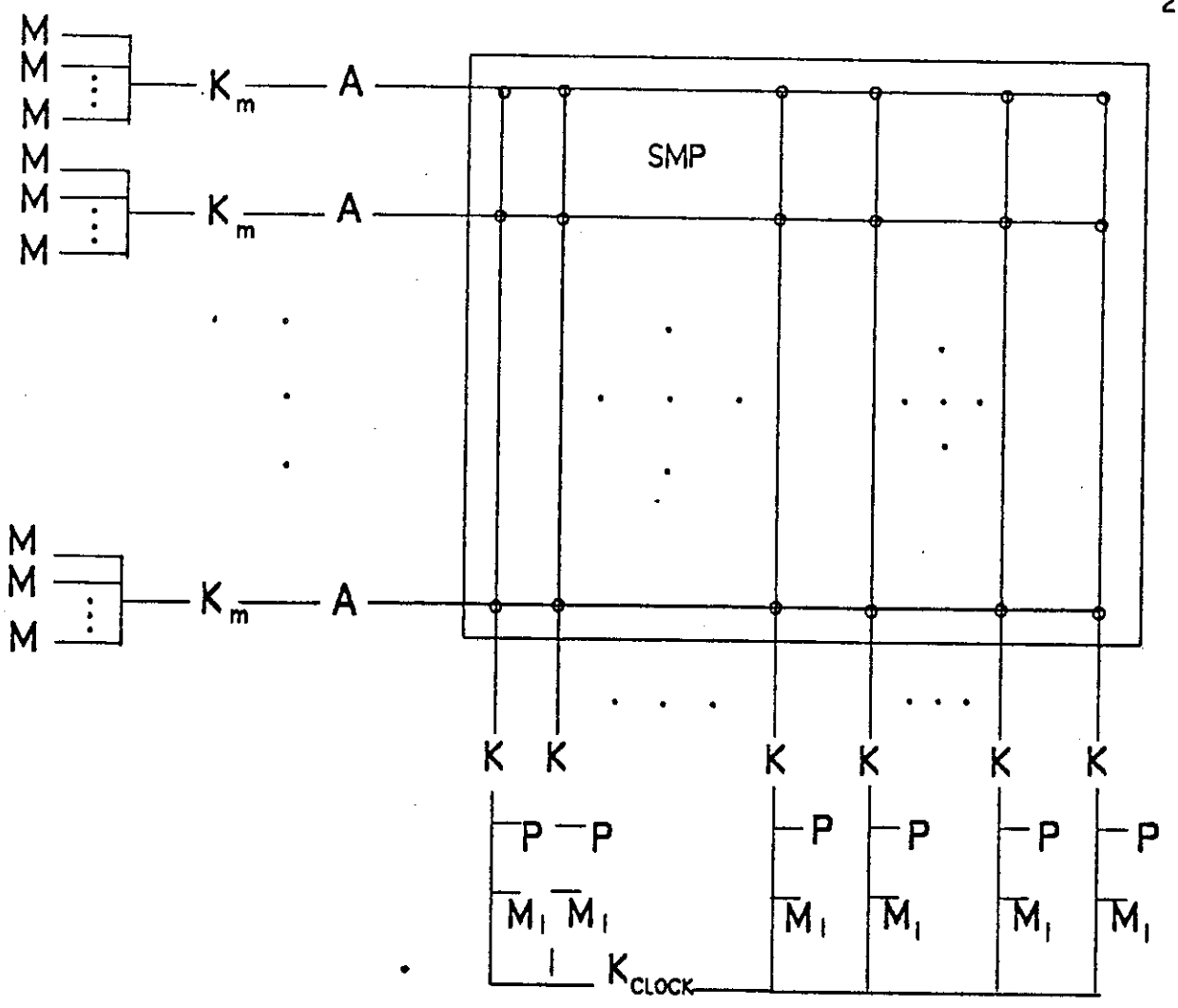


Figure 7. C.mmp reliability model.

Module	Minimum number
Processor	n
Local memory module/processor	v
Processor controller	n
Master clock	1
Shared memory module	m
Memory port controller	$\lceil \frac{m}{n} \rceil$
Memory port arbiter	$\lceil \frac{m}{n} \rceil$
Switch	1 (if lumped)
	$n * \lceil \frac{m}{n} \rceil$ (if distributed)

Table 9. Minimal module requirements
m,n,v functions of application requirements, rest derived

R_{pr}	Reliability of processor.
R_{pk}	Reliability of processor port controller.
R_{lm}	Reliability of local memory module.
R_{mk}	Reliability of memory port controller.
R_{ma}	Reliability of memory port arbiter.
R_{sm}	Reliability of shared memory module.
R_{sc}	Reliability of switch cross point.
R_{sl}	Reliability of lumped switch.
R_{mc}	Reliability of master clock.

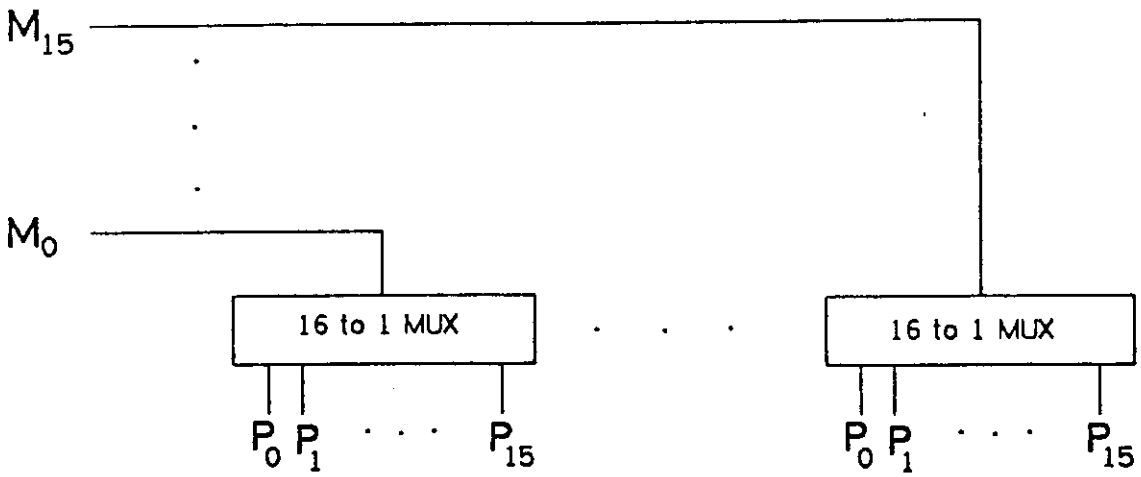
Table 10. Reliability parameters.

EQUIPMENT	CHIPS	GATES	BITS	λ/Mhr
PDP-11/40 (each of 16)				
Processor (pr)	416	3442	15872	56.933
CMU modifications to Processor (pk)[1]	53	492	0	7.610
Relocation Box (each of 16) (pk)	99	736	768	12.744
Memory				
EMM box 32Kw (sm) (each of 32, 2/port,byte parity)	189	413	589,824	159.638
EMM interface (per EMM box) (sm)	42	516	0	6.918
Memory Control (per port) (mk,ma)	20	197	0	2.918
Local Clock (pk) (each of 16; one per processor)	116	1504	256	18569
Master Clock (one only) (mc)	83	1717	0	14.704
16x16 Crosspoint Switch (sl)				
Crosspoint logic (total)	1656	29808	0	328.923
Priority Decode logic (total)	864	7344	0	121.664
Processor Interface logic (total)	384	3552	0	57.104
Front Panel, (Crosspoint Enable/Disable etc., total)	544	4448	0	77.712

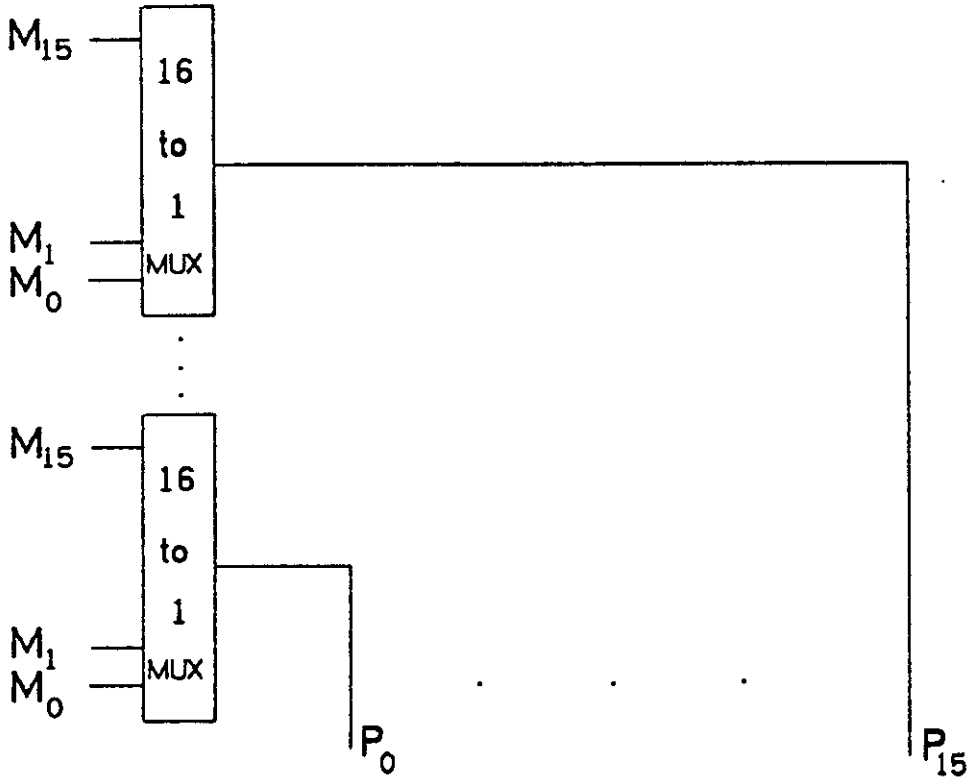
Table 11. Complexities and predicted failure rates for modules in C.mmp

The switch was modeled in two ways. The simplest case (lumped) is when the switch is considered to be a single component whose failure causes the whole system to fail. The second case more accurately reflects the construction of the switch. Figure 8 shows a single bit slice of the actual switch implementation. It is obvious from this Figure that a 16 by 16 switch is not made of 256 (16*16) individual cross points, but rather is 16 sets of cross points from one processor port to the sixteen memory ports, and another 16 sets of cross points from one memory port to the sixteen processor ports. Thus the failure rate of the components used as the communication path from a processor port to all the memory ports was added to the processor port controller failure rate, and the failure rate of all the components used as the communication path from a memory port to all the processor ports was added to the memory port controller failure rate.

[1]The lower case letters within parentheses in this table refer to the subscripts on the Reliability Parameters in Table 10 e.g. R_{pk} is assumed to be an exponential with a failure rate which is the sum of the failure rates on lines which have the annotation "(pk)" in this table.



a.)-Processor to memory path



b.) Memory to processor path

Figure 8. Physical implementation of crosspoint switch.

The system reliability R_{sys} can be stated as a function of the aggregate reliability of the processor and the aggregate reliabilty of the memories. We write

$$R_{sys} = R_{mc} * \sum_{i=0}^{P-n} \binom{P}{i} R_{AGP}^{P-i} (1-R_{AGP})^i R_1$$

where

$$R_{AGP} = R_{pk} * R_{pr} * \sum_{j=0}^{N-v} \binom{N}{j} R_{lm}^{N-j} (1-R_{lm})^j$$

and

$$R_1 = \sum_{k=0}^{MPD} \binom{T}{k} R_{AGM}^{T-k} (1-R_{AGM})^k R_2$$

where

$$MPD = T - \left\lceil \frac{m}{M} \right\rceil$$

= Maximum number of memory ports that can fail because of memory port arbiter or controller failure.

$$R_{AGM} = R_{mk} * R_{ma}$$

= overall reliabilty of memory port controller and memory port arbiter.

and further, R_2 gives the reliability of the memories when m of them are required for the application.

$$R_2 = \sum_{A=0}^{(T-k)M-m} R_{sm}^{(T-k)M-A} (1-R_{sm})^A R_{lmp}$$

$$R_{lmp} = \sum_{\alpha=t}^{MMP} B_\alpha R_3$$

$$MMP = Min \left[\left\lfloor \frac{A}{M} \right\rfloor , T - k - \left\lceil \frac{m}{M} \right\rceil \right]$$

$$B_\alpha = \binom{T-k}{\alpha}\left[\binom{(T-k-\alpha)M}{A-\alpha M} - \sum_{h=1}^{\gamma}\left[\binom{T-k-\alpha}{h} A_{\alpha+h} / \binom{T-k}{\alpha+h}\right]\right]$$

$$\gamma = \left\lfloor \frac{A}{M} \right\rfloor - \alpha$$

where B_α = number of cases such that failure of 'A' memory modules causes α memory ports to go down.

$$t = \begin{cases} 0 & \text{if } \left\lfloor \frac{A}{(T-k)(M-1)+1} \right\rfloor = 0 \\ (A-(T-k)(M-1)) & \text{otherwise} \end{cases}$$

Let D = T-k-α
= number of memory ports functioning
= number of switch cross points working per processor port.

If for simplification we assume that each processor should have access to all the required shared memory modules at any given time then

$$R_3 = \sum_{r=0}^{P-i-n} \binom{P-i}{r} \sum_{g=0}^{\rho} \binom{D}{g} (P-i-r)\, R_{sc}^{(P-i-r)(D-g)} * (1-R_{sc})^{(P-i)g+r(D-g)}$$

$$\rho = D - \left\lceil \frac{m}{M} \right\rceil$$

where R_{sc} is the reliability of a switch crosspoint. In the case of a lumped switch

$$R_{lmp} = R_{sl} * \binom{(T-k)M}{A}$$

Table 11 lists the major modules in the C.mmp implementation with their complexity and lumped failure rate. These failure rates were used in the above reliability model to generate the curves in Figures 9,10,11,12 and 13. C.mmp was modelled assuming that all processors were PDP-11/40's. One million words of semiconductor memory distributed 64Kw per port was also assumed. Figures 9 illustrates the system reliability (lumped switch) for various values of required processors (e.g. 4,8,12, and 16, assuming only 50% of the shared memory

is required). Comparing the curves for four and eight required processors indicates that the extra four processor spares contribute little to increasing system reliability. Figure 10 shows even less sensitivity for the 4, 8, and 12 required processors curves since the requirement for 75% of the initial memory is dominating the extra reliability contribution of the spare processors (lumped switch). Figure 11 shows the system reliability for various numbers of required processors with a distributed switch. In this model the failure rate of each set of 16 cross points which cause a memory (processor) port to go down is added to the memory (processor) controller failure rate. The comparison of this curve with Figure 9 (lumped switch, same configuration and same requirements) reveals that a lumped switch is a critical resource of relatively high failure rate with respect to other modules in the system. Of course, as was mentioned before, Figure 9 (lumped switch) is not an accurate model of the hardware because in actuality all switch failures do not cause the whole system to fail. Figure 11 more accurately reflects the reliability of C.mmp and also illustrates the effect of more detailed reliability modeling of critical resources. For example the MTI (Mission Time Improvement), the time for which the system reliability (R_{sys}) is above a certain minimum mission reliability [10], of the distributed switch over the lumped switch at a reliability of 0.9 for eight required processors (Figures 9, and 11) is 2700/350, or about 7.7.

Figure 12 illustrates the system reliability (distributed switch) for various values of memory modules (e.g. 256Kw, 512Kw and 768Kw memory and assuming only 12 processors are required.) The difference between the curves for 256Kw and 512Kw of required memory shows that the extra redundant memory adds little to system reliability.

Figure 13 illustrates the effect of using high reliability components. The architecture configuration parameters and the minimum requirements for the two systems plotted are the same except that the failure rate of all the modules in the system with high reliability components is assumed to be a factor of ten lower than the other system (representing higher level of component screening and hence a smaller value of π_Q in Mil 217B). The impact of high reliability components on the system is considerable.

3.4 Cm*, a Modular Multi Microprocessor

3.4.1 Architecture summary

Cm* is a modular multiprocessor system based on the LSI-11 processor. Each computer module (Cm) is connected via an interface (S.local) to an intelligent cluster controller, K.map. The clusters of Cms can be interconnected via Linc's and intercluster buses. Each Cm can share memory with any other Cm in the network through routing tables in the K.map. The S.local controls local memory access and passes external refrences (i.e. to memory elsewhere in the cluster or in a different cluster) to the K.map which does the appropriate mapping and routing of the request. The K.map enforces a capability based protection scheme and provides considerable support for operating system primitives and interprocess communication. The architecture provides for incremental extensibility, modularity, reliability, and an effective cost/performance ratio. The companion paper [1] in this issue provides a more detailed description of the Cm* architecture.

3.4.2 Probabilistic hard failure model for Cm*

Due to the flexibility of the Cm* architecture, two particular configurations were selected for comparison. Figure 14 shows the case where there are two parallel intercluster buses and all the clusters are connected to these two buses via Linc's. Figure 15 shows the case

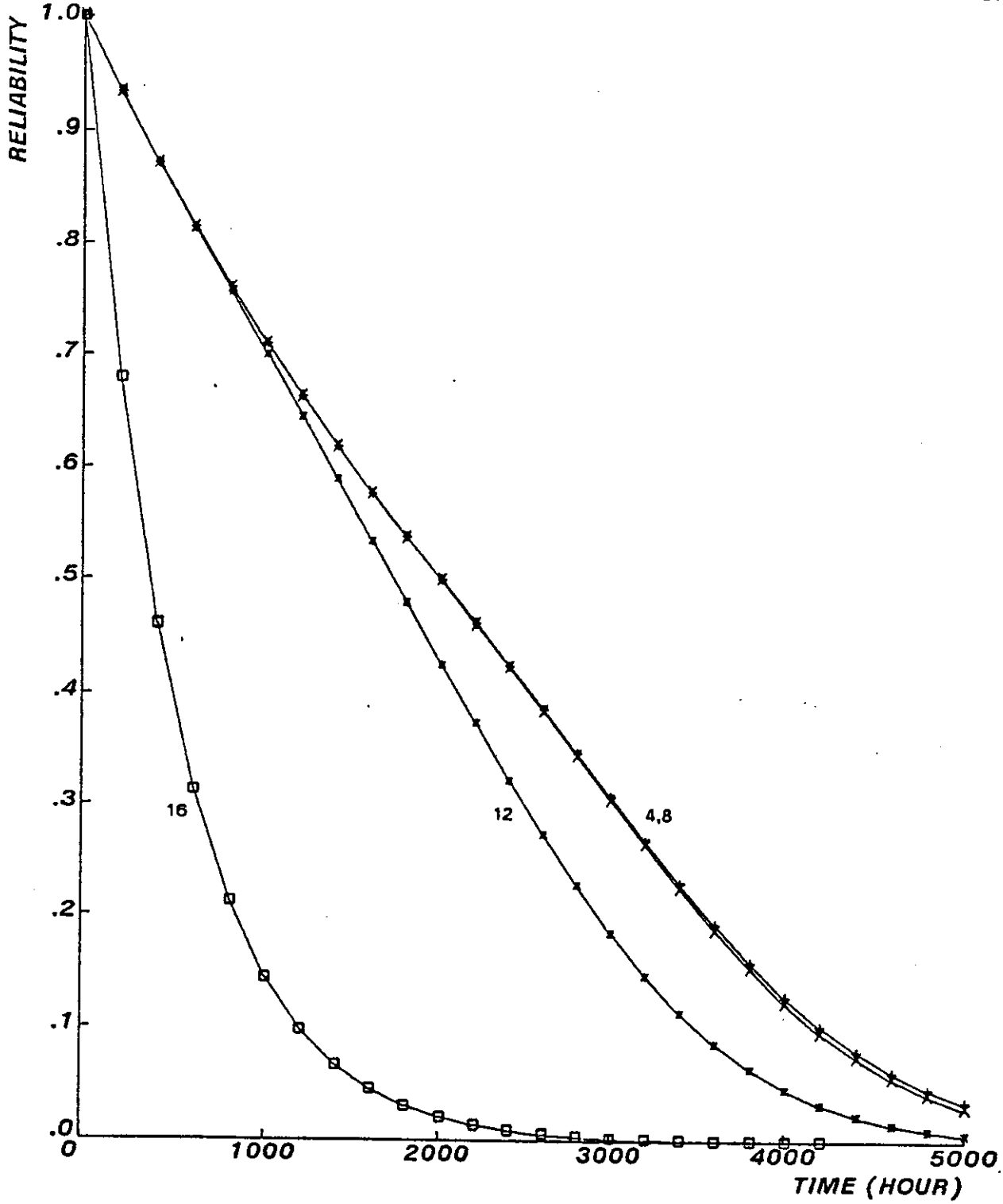


Figure 9. C.mmp with 64Kw per port, 512Kw required (lumped switch)

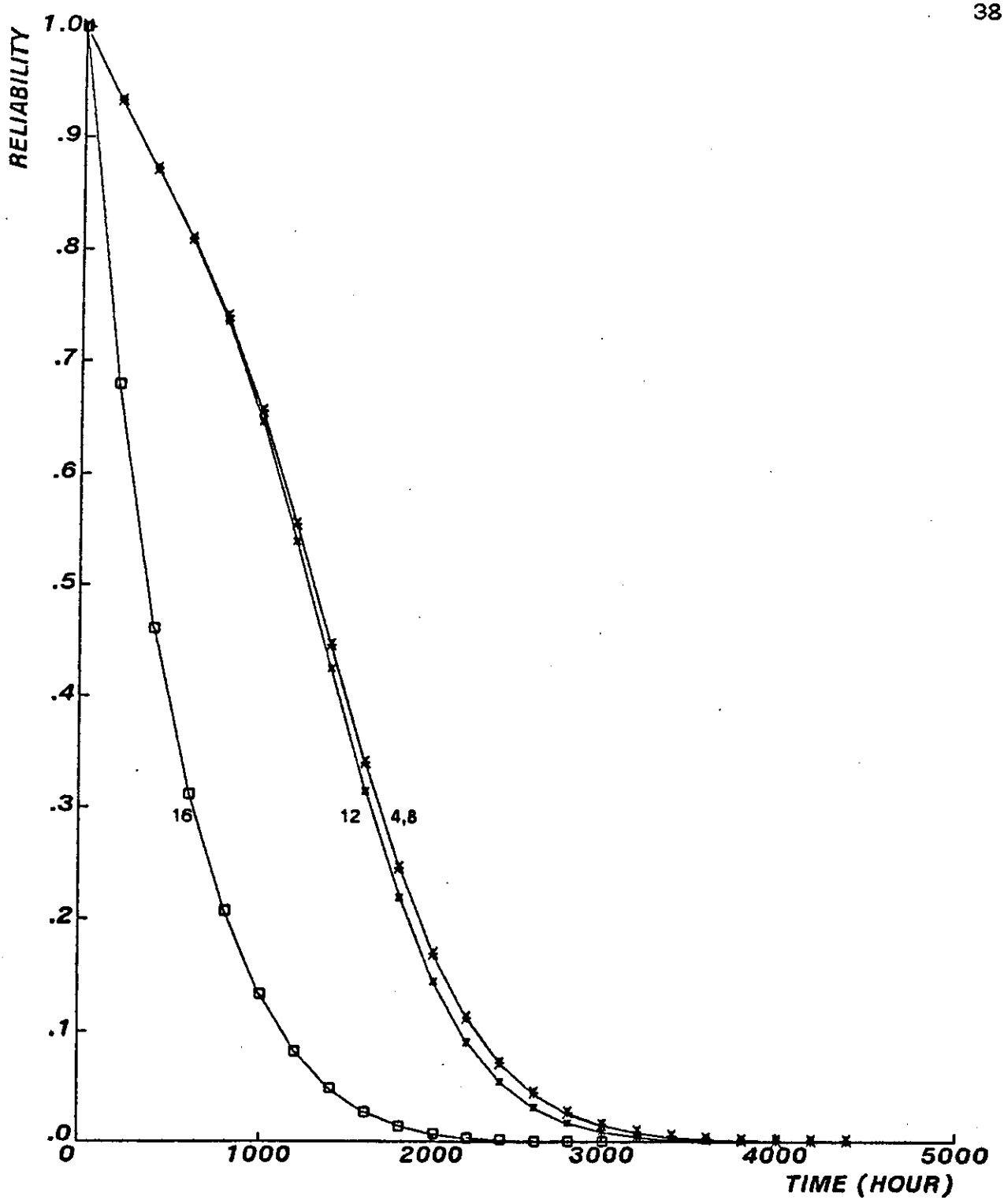


Figure 10. C.mmp with 64Kw per port, 768Kw required (lumped switch)

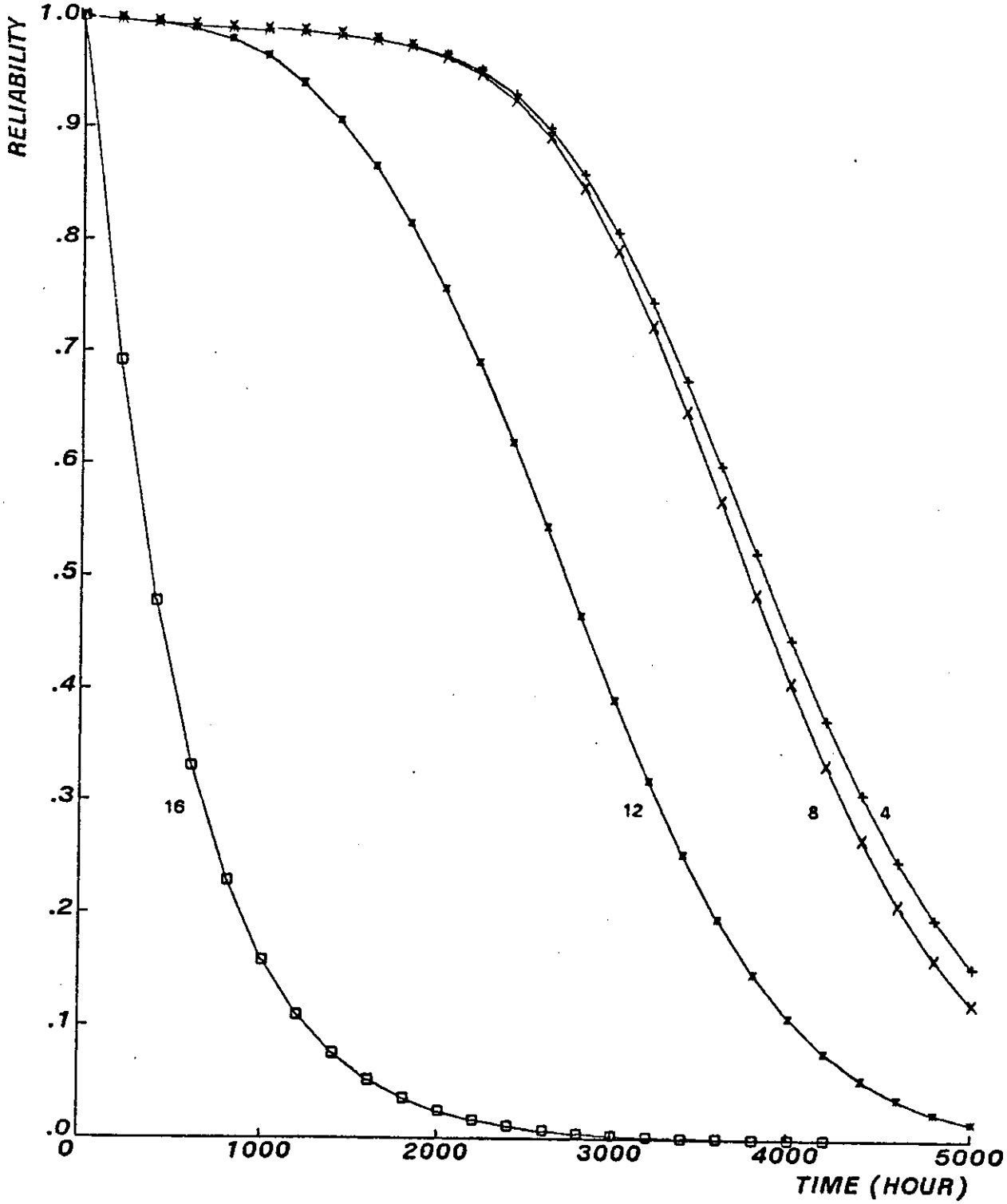


Figure 11. C.mmp with 64Kw per port, 512Kw required (distributed switch)

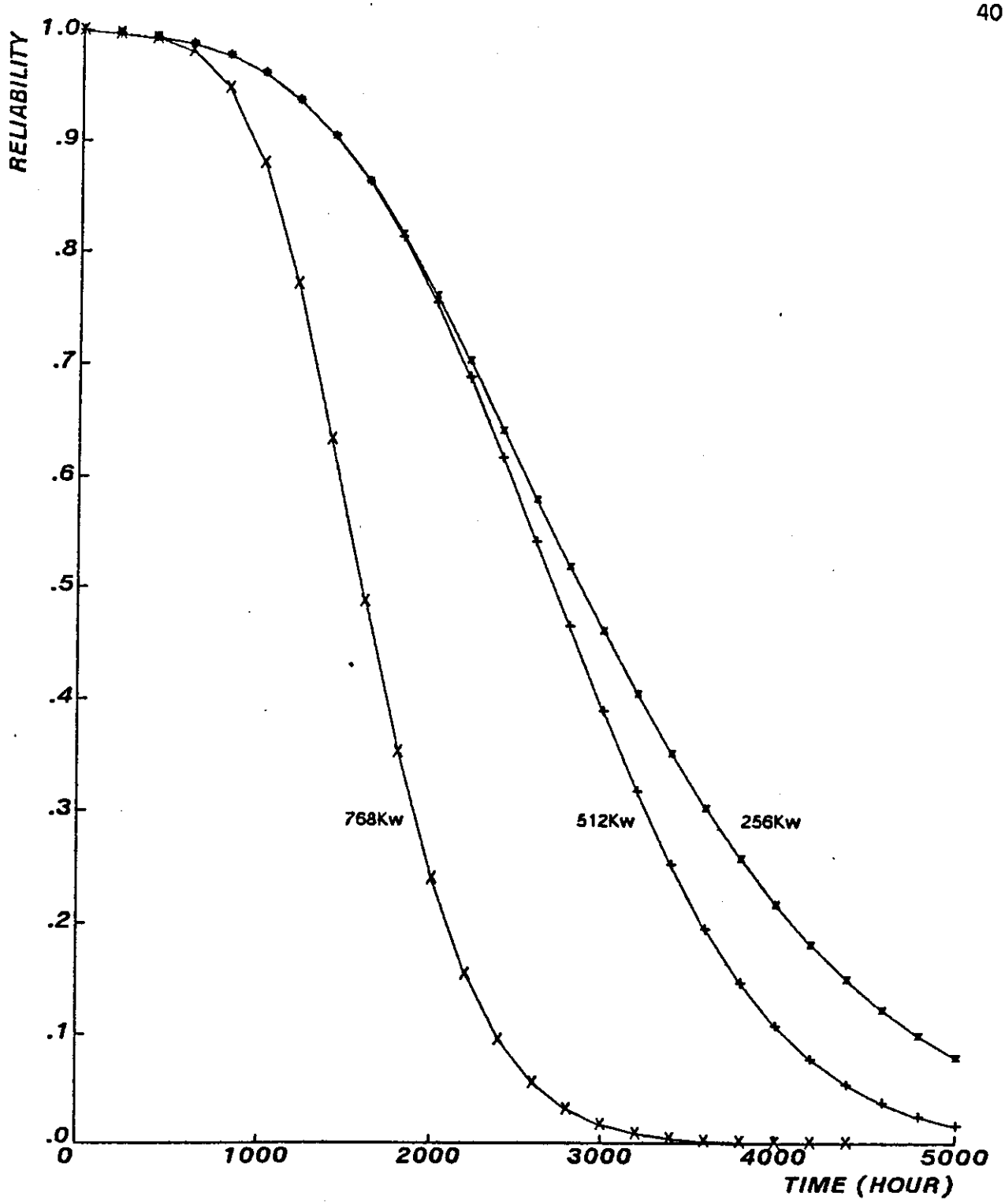


Figure 12. C.mmp with 64Kw per port, 12 processors required (distributed switch)

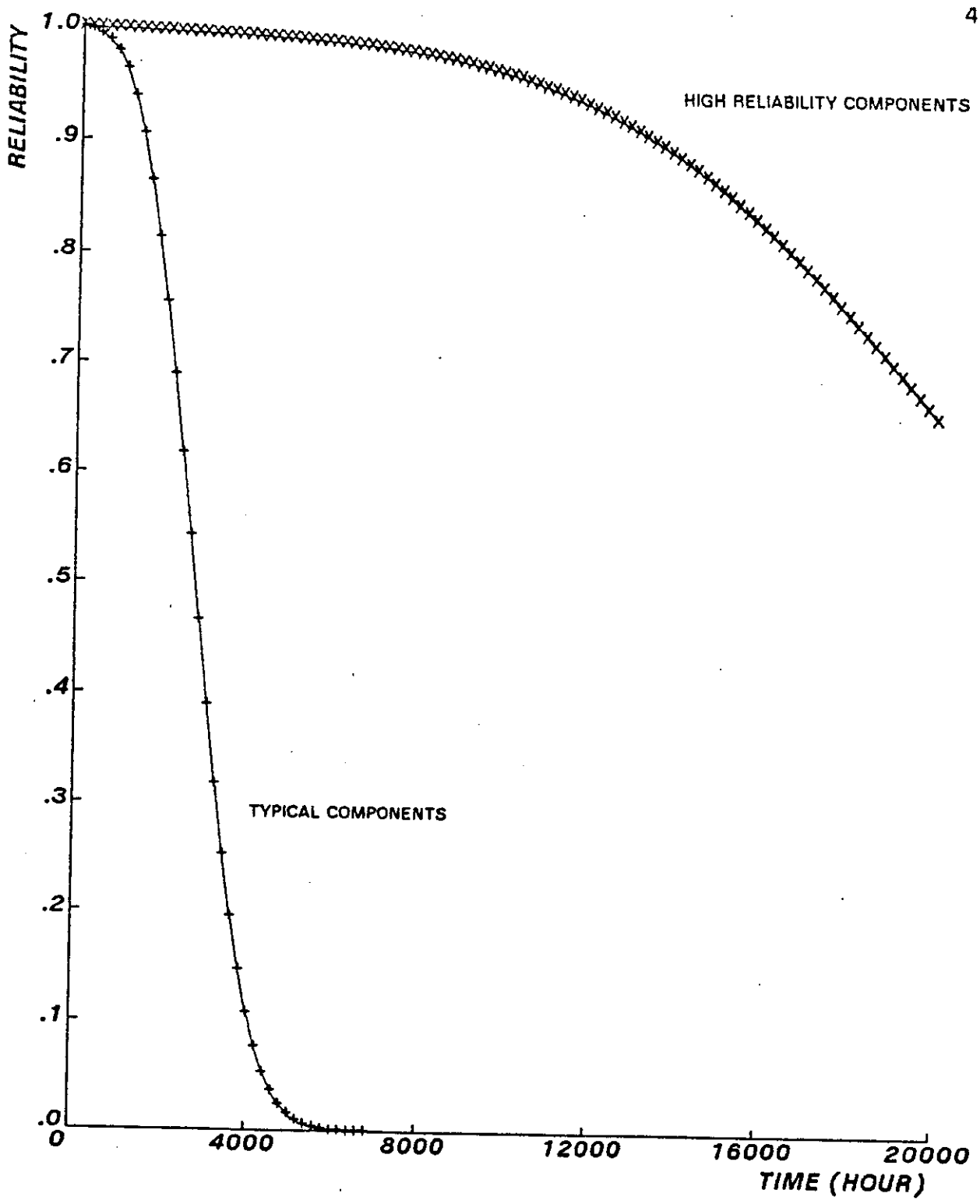


Figure 13. C.mmp with 64Kw per port, high reliability components, 12 processors and 512Kw required (distributed switch)

where the buses form a checkerboard pattern, again each cluster is connected to two of the buses via two Linc's. No two clusters are connected to the same two buses in this configuration. Table 12 depicts the effect of module failure in Cm*, Table 13 lists the architectural parameters, Table 14 shows the minimal module requirements, Table 15 lists the reliability parameters, Table 16 summarises the failure rates for the modules as calculated by AUTOFAIL.

Module	Effect of failure
Processor	Loss of processor,
Memory module	Loss of memory module.
S.local	Loss of CM,
K.map	Loss of cluster,
Linc	Loss of link, isolation of cluster in the case of two Linc failures.
Inter cluster bus	Loss of bus, perhaps separation of clusters.

Table 12. Effect of module failures in Cm*

C Number of clusters
P Number of processors per cluster.
M Number of memory modules per CM.
B Number of intercluster buses.

Table 13. Architectural parameters

Module	Minimum number
Processor	n
Memory module	m
S.local	$MS = Max\left[n, \lceil \frac{m}{M} \rceil \right]$
K.map	$Max\left[\lceil \frac{n}{p} \rceil, \lceil \frac{m}{M*P} \rceil \right]$
Linc	$Max\left[\lceil \frac{n}{p} \rceil, \lceil \frac{m}{M*P} \rceil \right]$
Bus	1 (two parallel bus) (depends on the structure)

Table 14. Minimal module requirements
(n,m a function of application requirements, the rest derived).

R_{proc}	Reliability of processor in Cm
R_{mem}	Reliability of a memory module
$R_{s.local}$	Reliability of S.local
R_{bus} [1]	Reliability of intercluster bus
$R_{k.map}$	Reliability of K.map
R_{linc}	Reliability of Linc
R_{qi}	Reliability of i clusters that can communicate

Table 15. Reliability parameters

[1] In all the calculations R_{bus} assumed to be 1.

EQUIPMENT	CHIPS	GATES	BITS	λ/Mhr
Computer Module				
LSI-11 with 4K word memory	69	7145	99,328	29.893
LSI-11 without 4K word memory (proc)	53	7145	33,792	12.902
4K word memory (mem)	56	438	65,536	23.139
Serial Line Unit	28	918	0	4.689
Power distribution	6	74	0	0.973
Memory refresh control	14	231	0	2.664
S.local (s.local)	126	3700	5376	24.059
K.map (k.map)				
Map bus controller, Kbus	138	8494	12,480	33.360
Mapping processor, Pmap	155	3892	2816	35.600
Segment descriptor memory	147	1081	92,160	35.454
Microcode memory	116	376	84,480	26.581
Linc (linc)	150	3443	32,960	34.836

Table 16 Complexities and predicted failure rates for modules in Cm*

Consider the three cluster checkerboard connected Cm* (Figure 15). It is evident that the checkerboard is not a regular structure, i.e. failure of modules of the same type do not have an equivalent effect on the system functionality. For example, failure of K.map number 1 leaves a two isolated cluster system, whereas failure of any of the two other K.map's (numbers 2 and 3) leaves a single two connected cluster system, etc. This irregularity complicates the generation of an analytical solution for reliability of large arbitrary Cm* structures. A program was written to enumerate all possible functional cases based on the states of the subset of modules that are required for inter-cluster communication (i.e. buses, K.map's and Linc's) in the Cm* architecture. For each of the cases, the closed form analytical solution for the resultant clusters was used to generate the contribution to reliability for that particular case. The reliability of the entire structure is then a combination of the reliability of these special cases. The number of cases enumerated in this fashion is very much less than the total number of functional states which the structure may occupy. At any stage of enumeration the available modules will be checked against the minimum requirements, if they do not satisfy the minimum requirements that case will not be considered further.

The system reliability will be evaluated with respect to the module failure effects, architectural parameters, minimal module requirements and reliability parameters in Tables 12, 13, 14, and 15.

To evaluate the reliability of the whole system we follow the example in Section 3.2, i.e. calculate the contribution to the system reliability of each working state. The overall reliability of the system will be sum of the contributions of all the working states.

The general formula for system reliability to be derived in this section is valid for any Cm* strusture (parallel bus, checkerboard pattern, etc). Assume after the failure of α buses, β

K.maps's and ϵ Linc's there exist f sets of intercommunicating clusters. Each set has qi connected clusters. Any of the parameters used in the following equations and not mentioned are previously defined in the tables.

$$\text{let } R_{qi} = \text{reliability of set i } \left(\sum_{i=1}^{f} qi = C \right)$$

Then the reliability of the whole system is

$$R_{sys} = \left[1 - \prod_{i=0}^{f} (1-R_{qi})\right] R_{bus}^{B-\alpha} (1-R_{bus})^{\alpha} R_{k.map}^{C-\beta} (1-R_{k.map})^{\beta} *$$

$$R_{linc}^{C-\epsilon} (1-R_{linc})^{\epsilon}$$

(It is assumed that each cluster is connected to two buses.)

where $R_{qi} = 0$ if set i does not satisfies the minimum requirements and

$$R_{qi} = \sum_{a=0}^{qi*P-MS} \binom{qi*P}{a} R_{s.local}^{qi*P-a} (1-R_{s.local})^{a} R_1$$

and

$$R_1 = \left(\sum_{b=0}^{qi*P-a-n} \binom{qi*P-a}{b} R_{proc}^{qi*P-a-b} (1-R_{proc})^{b} \right) *$$

$$\left(\sum_{e=0}^{(qi*P-a)*M-m} \binom{(qi*P-a)*M}{e} R_{mem}^{(qi*P-a)*M-e} (1-R_{mem})^{e} \right)$$

The further evaluation of R_{sys} is a function of the particular Cm* structure. Consider the checkerboard structure with identically configured clusters and an application can be met by a single cluster. Then there are six possible major states of the configuration in Figure 15 which provide at least one functioning cluster. They are:

1. All components function

2. All components function except at least one Linc is failed

3. Both Intercluster buses function, all four Lincs function, one or two K.map's are

failed

4. Both Intercluster buses function, one or two K.map's are failed, at least one Linc is failed

5. At least one Intercluster bus is failed, all K.map's and all their Lincs function

6. At least one Intercluster bus is failed, one or two K.map's are failed.

Note that each of these cases consists of several sub-cases. The system reliability R_{sys} is the sum of the reliabilities in each of the six cases above. As an example, we shall only derive the equations for Case 2 in which all the components function except at least one Linc which is failed. The other cases can be derived in an analogous manner. With the notation of Table 15 we have the following sub-cases:

a) All components work except exactly one Linc

$$R = R_{bus}^2 * R_{k.map}^3 * R_{linc}^3(1-R_{linc}) *$$

$$\binom{4}{1} \left[1-(1-R_{q2})(1-R_{q1})\right]$$

The failure of exactly one Linc implies that at least two clusters will still be able to communicate while the third is isolated. The term within "[]"s is the probability that the task's processor and memory requirements are met either by two communicating clusters (Inclusive) OR by one isolated cluster.

b) All components function except exactly two Lincs

$$R = R_{bus}^2 * R_{k.map}^3 * R_{linc}^2(1-R_{linc})^2 *$$

$$\left\{2\left[1-(1-R_{q2})(1-R_{q1})\right]+\left(\binom{4}{2}-2\right)*\left[1-(1-R_{q1})^3\right]\right\}$$

Two Lincs can fail in a symmetric fashion (Lincs 1 and 3 or 2 and 4 in Figure 15) or in an asymmetric fashion (Lincs 1 and 2 or 3 and 4). In the former case we have one isolated cluster and two communicating ones. This is reflected by the first term in "[]"s within "{}"s, in the previous equation. In the latter case we have three isolated clusters. This gives rise to the second term in "[]"s within "{}"s, in the previous equation.

c) All components function except exactly three Lincs. There will be three isolated clusters.

$$R = R_{bus}^2 * R_{k.map}^3 * \binom{4}{3} R_{linc}(1-R_{linc})^3[1-(1-R_{q1})^3]$$

d) All components function except exactly four Lincs. Again all three clusters are isolated.

$$R = R_{bus}^2 * R_{k.map}^2 * (1-R_{linc})^4[1-(1-R_{q1})^3]$$

The reliabilities in each of the other five cases may be calculated in a similar way. The above model was programmed using predicted failure rates from the modified MIL Hdbk 217B model described in Section 2. Some of these failure rates are summarized in Table 16. Curves were plotted for the reliability of the Cm* model configurations for various values of p and m.

The configuration parameters used in plotting the curves are for a three cluster model Cm*. Each cluster has three Cm's and each Cm has a total number of 8, 4Kw memory modules.

Figure 16 illustrates reliability of a system with two parallel buses for various values of required processors (e.g. 3, 6, and 9, assuming 8, 4Kw memory modules per Cm and only 144Kw of memory required).

Figure 17 illustrates the reliability when all of the memory modules (288Kw) are required. This curve shows no improvement due to redundant processors, because memory has dominated the reliability contribution of the spare processors.

Figure 18 depicts the reliability of a system equivalent to that of Figure 16 but using a checkerboard pattern for the intercluster buses. A careful look at these two curves shows that the two parallel bus system is slightly more reliable. Figure 19 plots a curve from each configuration (i.e. the one with six processors required) on the same graph.

Figure 20 shows the case when six processors and 144Kw or 288Kw of memory are required. A 50% redundacy in memory yields a substantial increase in system reliability.

The configuration for a Cm* system for optimum reliability for a given set of minimum requirements has not yet been determined. The effect of interconnection structure on system reliability is ongoing research.

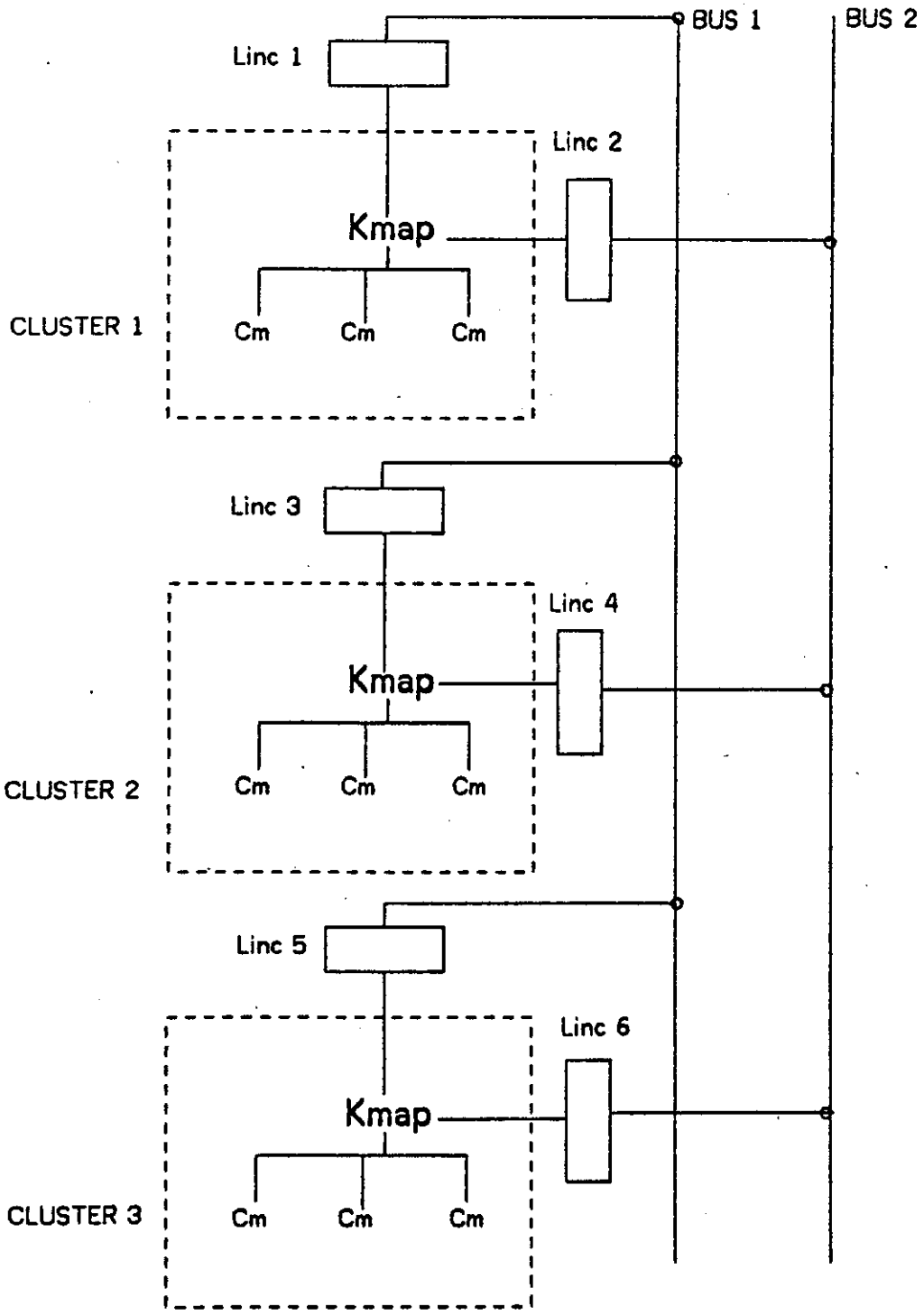


Figure 14. 3-Cluster Cm* configuration, two parallel bus pattern

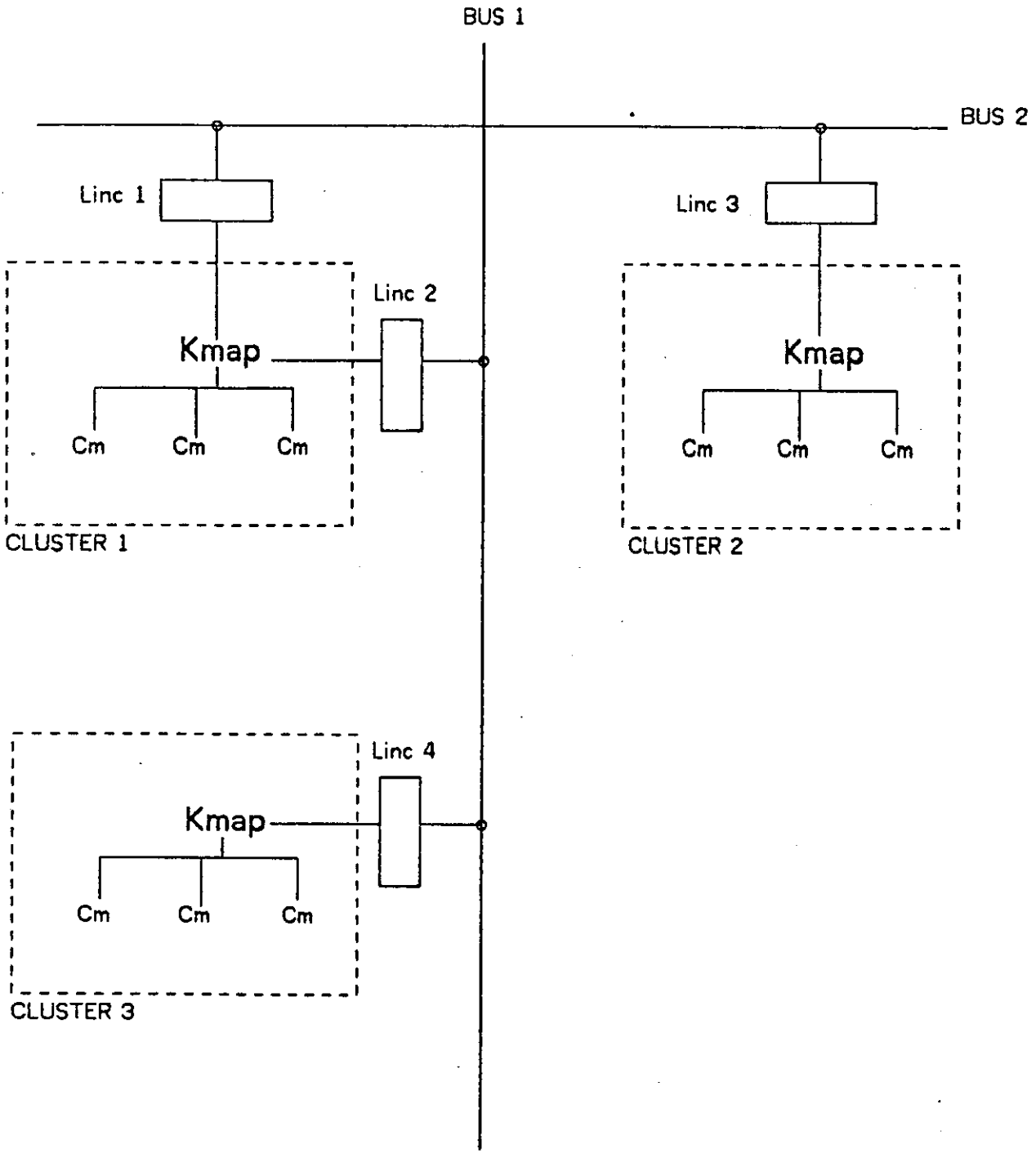


Figure 15. 3-Cluster Cm* configuration, checkerboard pattern

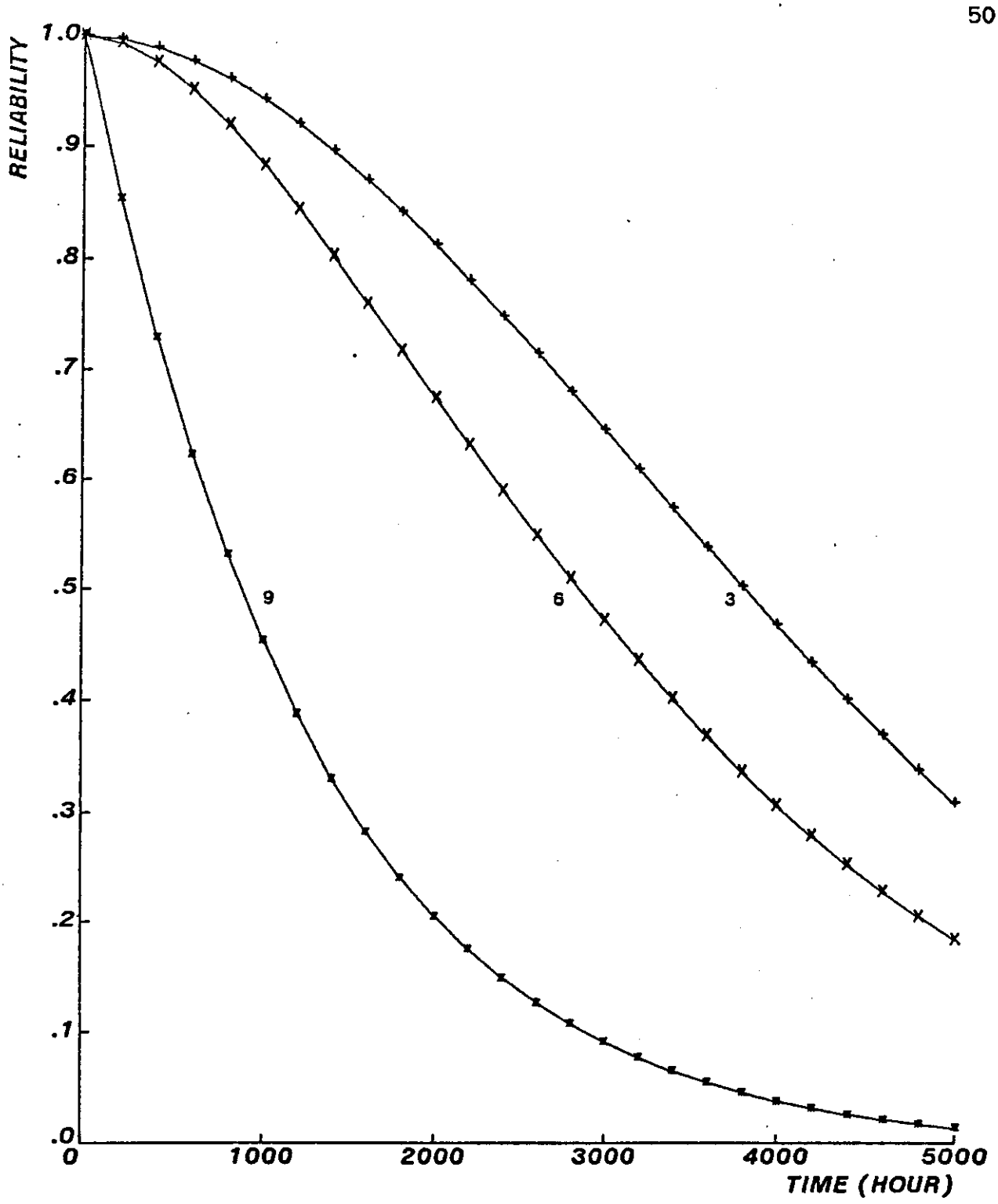


Figure 16. 3-Cluster Cm* with three processors/cluster,
32Kw/Cm, 144Kw required (parallel buses)

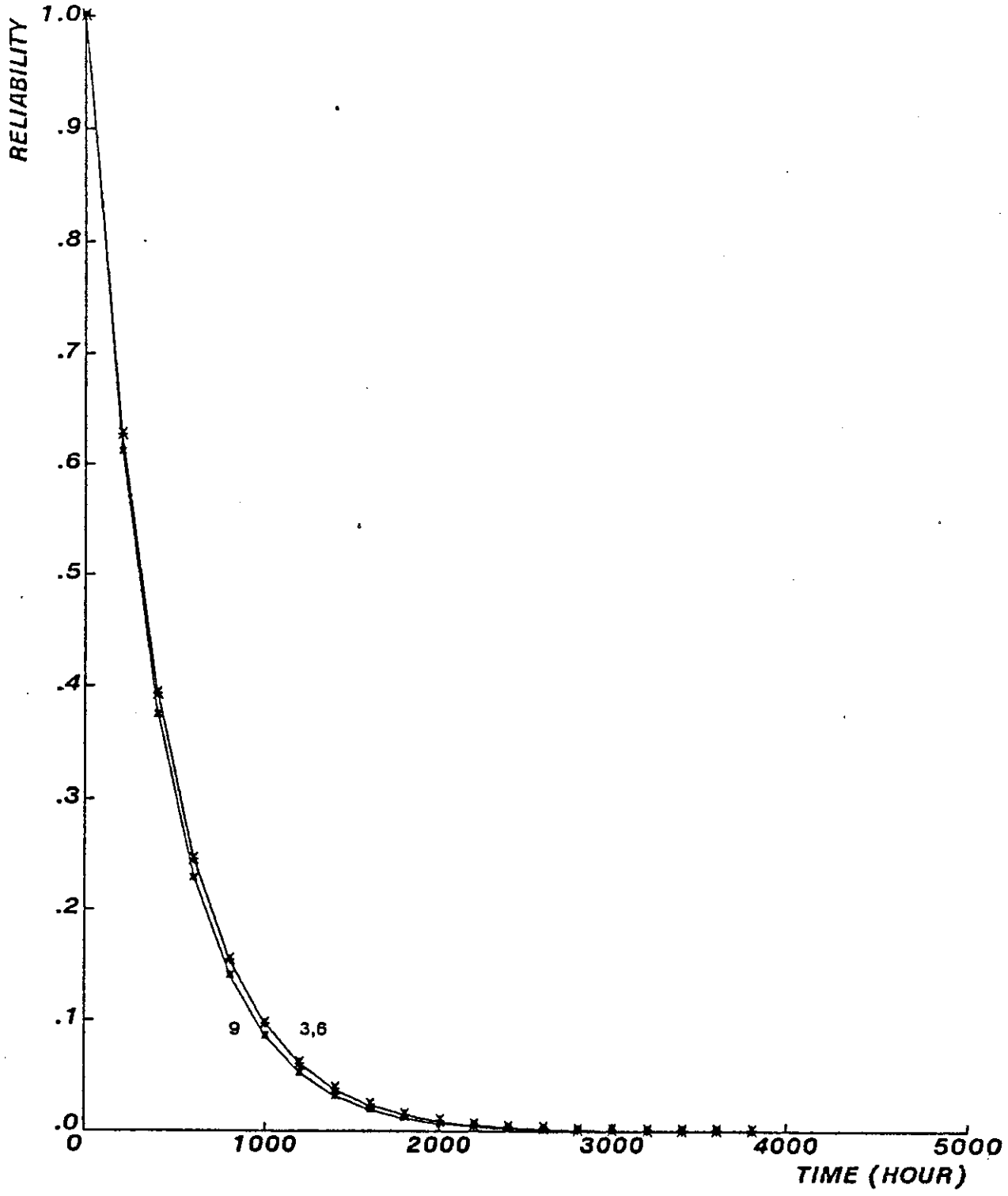


Figure 17. 3-Cluster Cm* with 3 processors/cluster, 32Kw/Cm, 288Kw required (parallel buses)

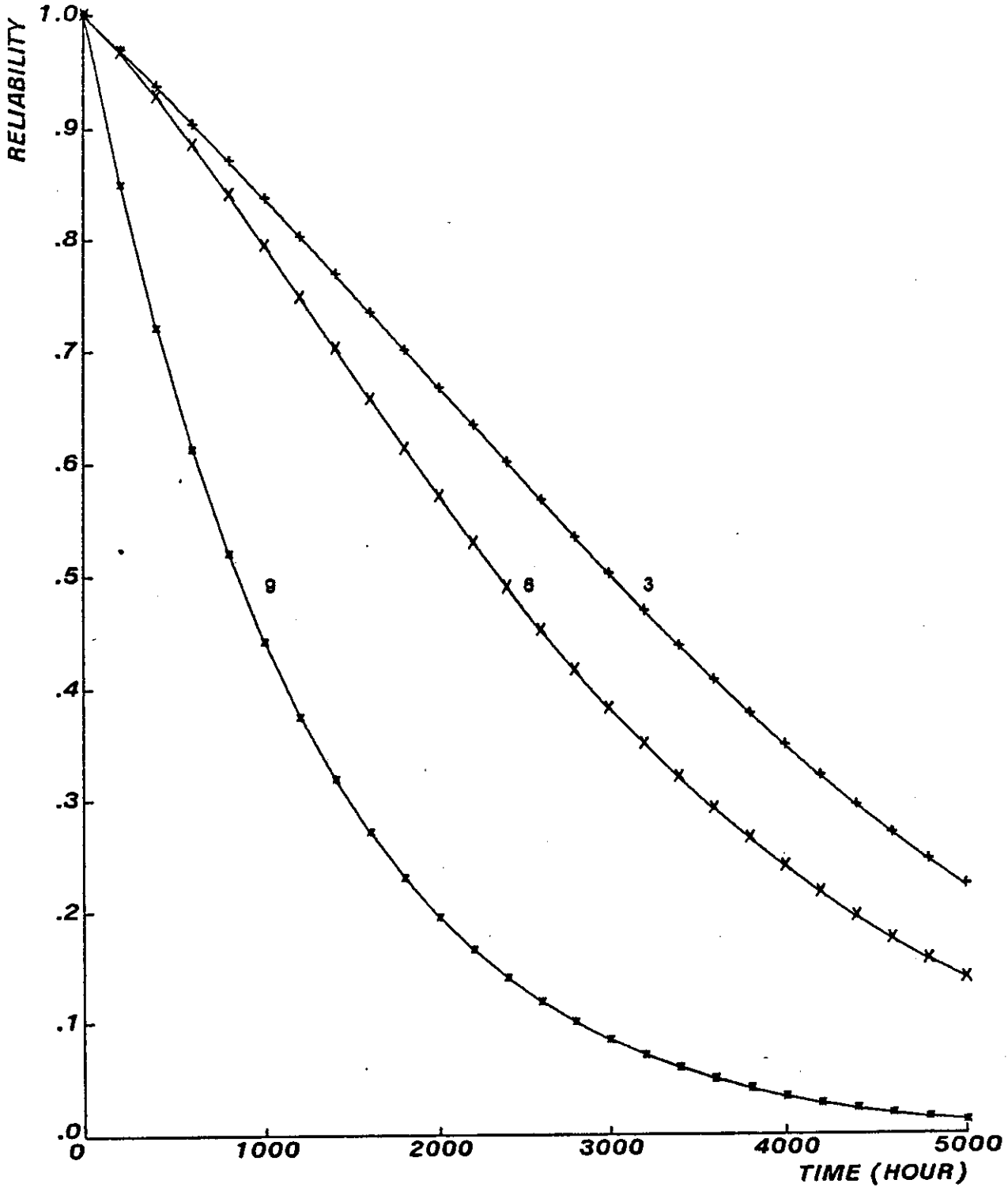


Figure 18. 3-Cluster Cm* with 3 processors/cluster,
32Kw/Cm, 144Kw required (checkerboard pattern)

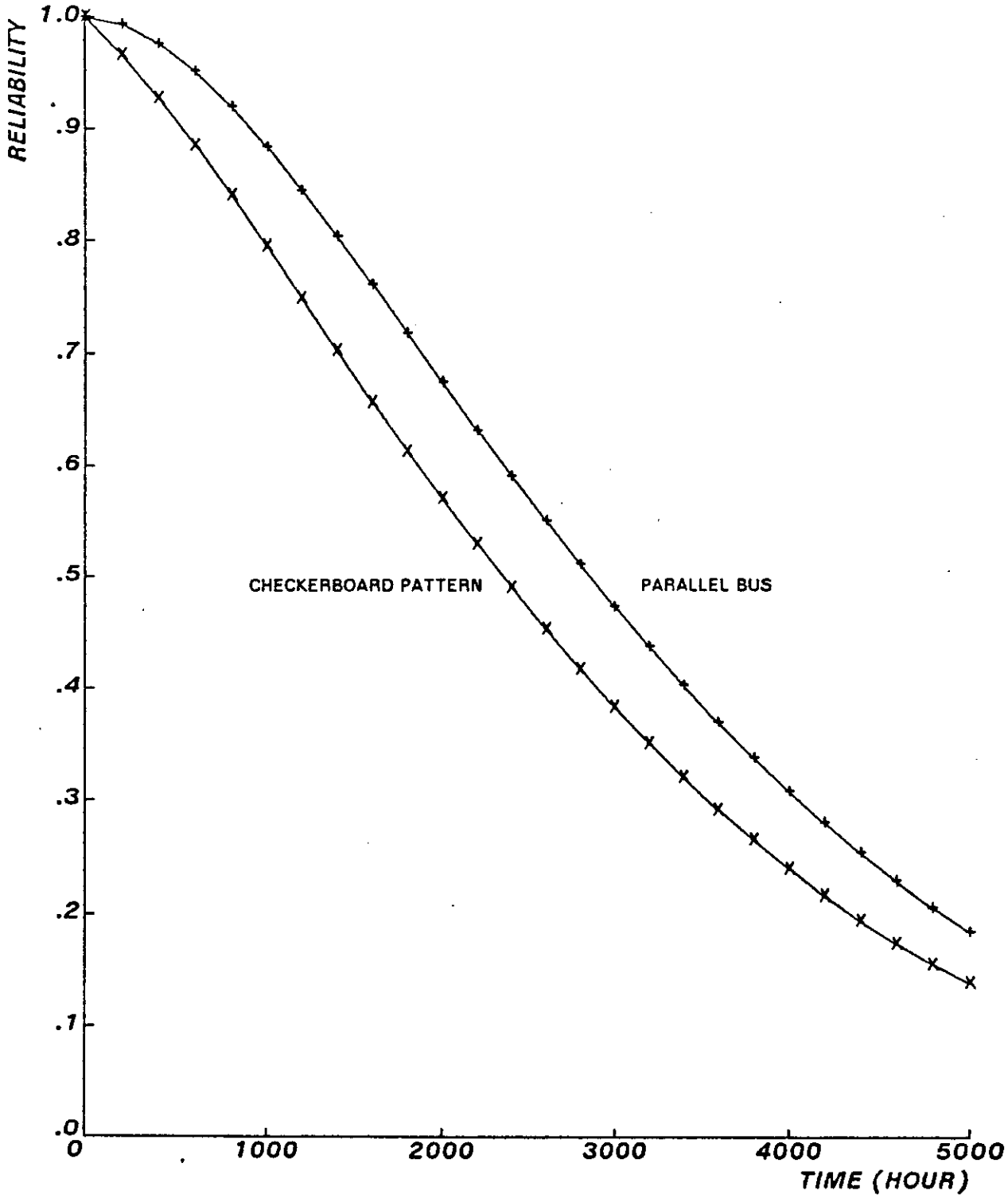


Figure 19. 3-Cluster Cm* with 3 processors/cluster,
32Kw/Cm, 144Kw and 6 processors required

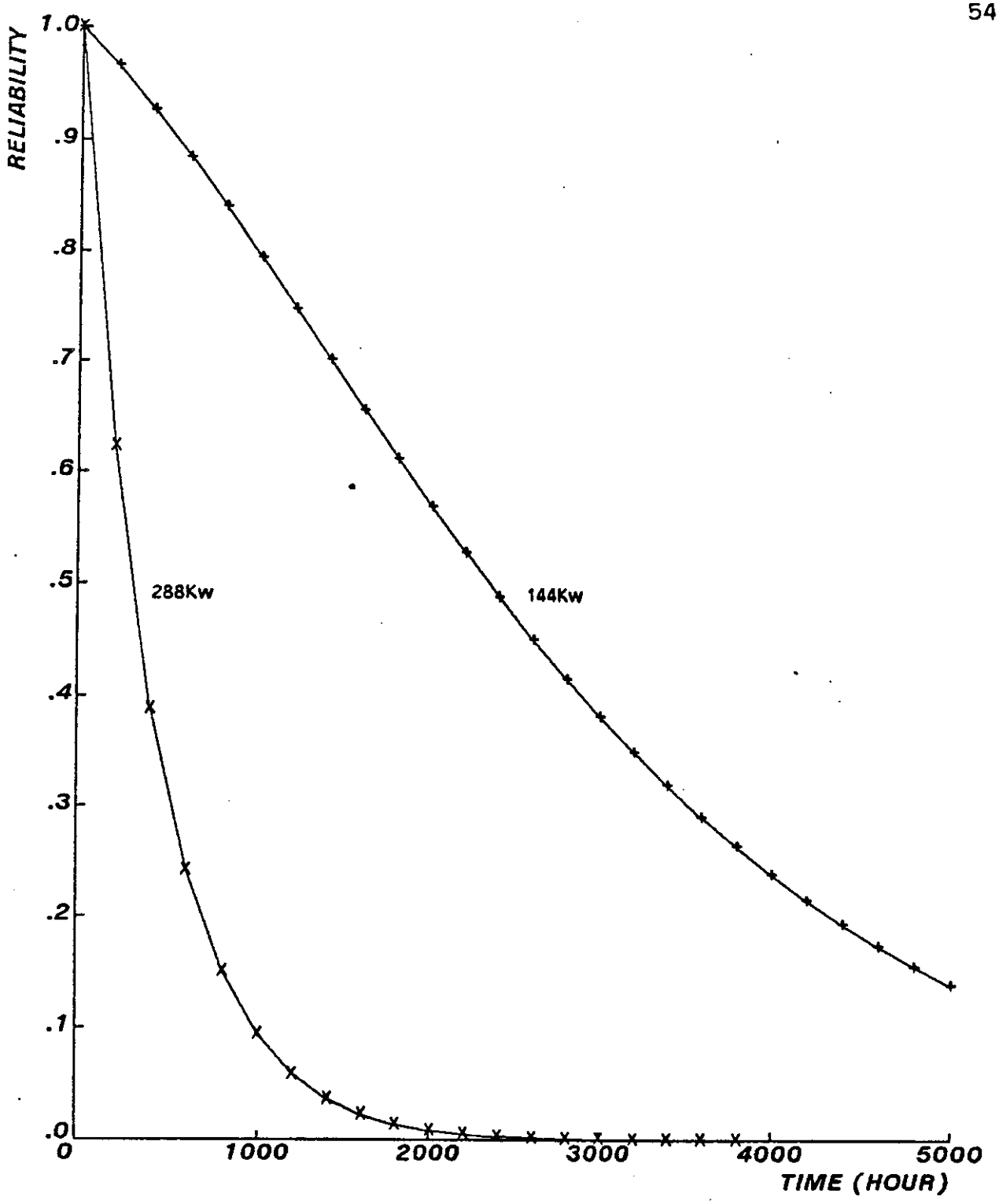


Figure 20. 3-Cluster Cm* with 3 processors/cluster,
32Kw/Cm, 6 processors required (checkerboard pattern)

3.5. C.vmp, A Voted Multiprocessor

3.5.1 Architecture Summary

C.vmp may best be described as a multiprocessor system capable of fault-tolerant operation (Figure 21). It consists of three separate LSI-11 microcomputers each with its own memory and peripherals. They may run independently as three separate computers communicating through parallel line units (L in Figure 21). Alternatively, they may be switched into what is termed voting mode under manual or program control to form a triplicated LSI-11. In this mode all three processors run identical programs operating on identical data. All signals at the bus level are voted upon by a majority voter in both the processor-to-memory direction as well as the memory-to-processor direction. This form of triple modular redundancy (TMR) allows the voted multiprocessor to continue operating under the situation where any one out of three copies of any triplicated element (e.g. processor, memory, floppy disk, bus line etc.) suffers a hard failure. Confining the voting to the bus level makes fault-tolerance transparent at the software level while allowing the use of off-the-shelf components. The capability to switch between voting and independent mode under program control allows dynamic tradeoffs between reliability and performance. A companion paper [1] in this issue provides a more detailed description of the C.vmp architecture.

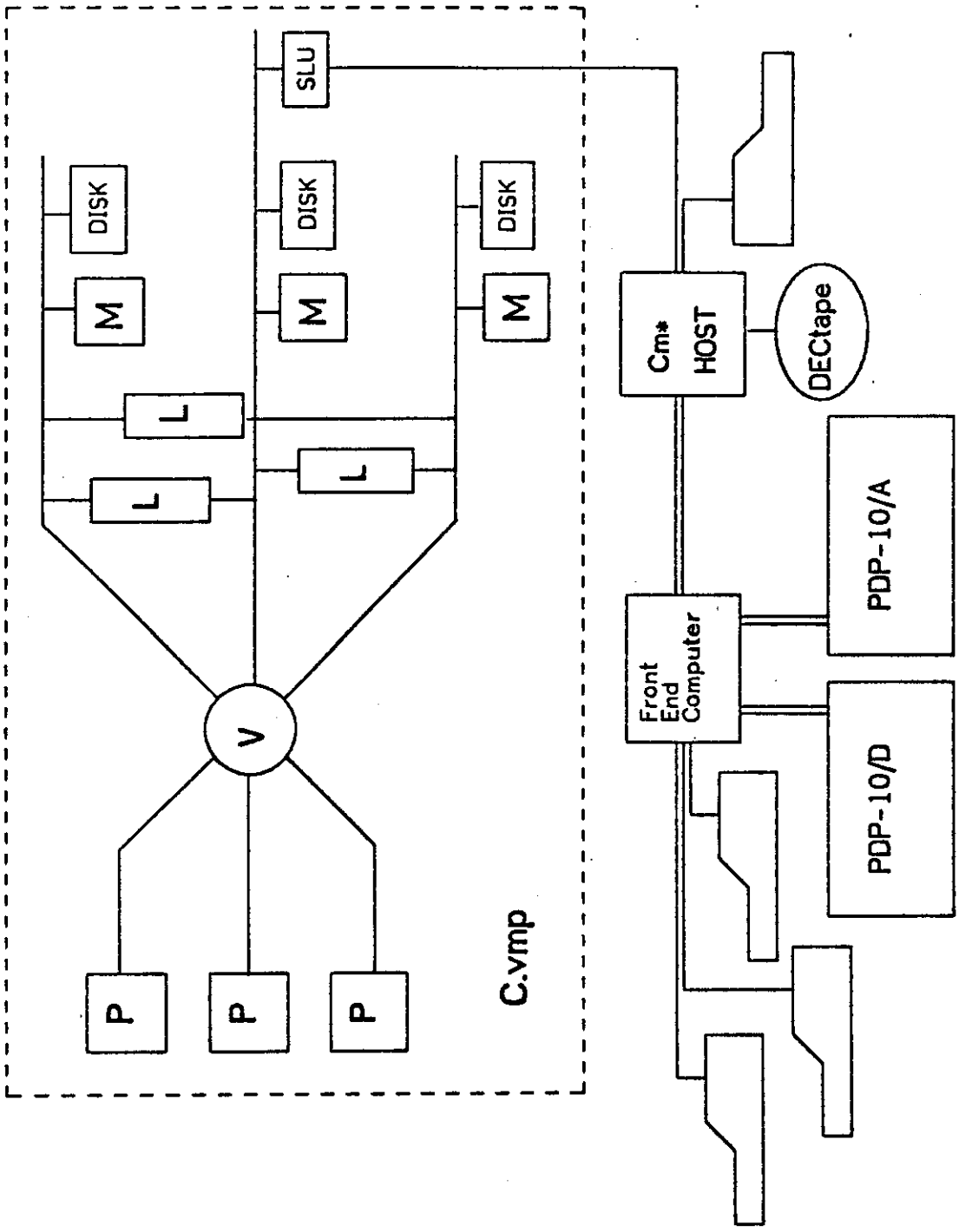


Figure 21. Architecture of C.vmp

3.5.2 Probabilistic Hard Failure Model for C.vmp

The effect of the failure of various modules and subsystems in C.vmp is shown in Table 17 and the reliability parameters are presented in Table 18. The reliability calculated for permanent faults is based on a parts count model of the system, using the AUTOFAIL program described earlier. For these calculations, we assume a hardware configuration for C.vmp consisting of three processors, 28K of memory per processor, and the voter. The resulting values are shown in Table 19. Table 20 summarizes the failure rates for the various modules for two different voter designs.

MODULE OR SUBSYSTEM	EFFECT OF FAILURE
Any single Processor	Loss of a Processor (System continues to function)
Any single Memory	Loss of a Memory (System continues to function)
Any single Voter bus interface	
Processor bus side	Loss of a Processor
Memory bus side	Loss of a Memory
both	Loss of a Processor and a Memory (system continues to function in each case)
Voter control circuitry	Loss of Voter (two Processors may function in independent mode)
Any one copy of any triplicated Bus line on Processor side or Memory side	Loss of a Processor or a Memory respectively

Table 17. Effect of module failures in C.vmp.

RELIABILITY PARAMETER	MEANING
R_p	Reliability of Processor
R_m	Reliability of single 4Kw memory module
R_v	Reliability of triplicated part of the voter
R_n	Reliability of non-triplicated part of voter
R_e	Reliability of the part of the voter associated with the external bus

Table 18. Reliability parameters.

EQUIPMENT	CHIPS	GATES	BITS	λ/Mhr
LSI-11 Processor (each of 3)	53	7145	33,792	12.902
4K RAM (each of 7 per processor)	56	438	65,536	23.139
Bus Level Voter: Non-Triplicated Version				
Control	22	189	0	3.165
Non-triplicated portion:	44	803	0	8.543
Triplicated portion (total for 3 buses)	143	1781	0	24.349
Bus Level Voter: Triplicated Version				
Control	22	189	0	3.165
Triplicated portion (total for 3 buses)	263	4107	0	48.424

Configurations:

Model 1: 3 LSI-11, 7x4Kw RAM/Processor, nontriplicated voter.

Model 2: 3 LSI-11, 7x4Kw RAM/Processor, triplicated voter.

Lumped Failure Rates:

Model 1: 560.682 /Mhr.

Model 2: 576.214 /Mhr.

Table 19. Complexities and predicted failure rates for modules in C.vmp

Three cases are considered:

> a nonredundant system consisting of only one processor with memory;

> a triplicated system with a nonredundant voter design; and

> a triplicated system with a redundant voter design--all devices in the data paths of the voter, including the voter chips themselves, are triplicated. It should be noted that a large portion of the non-redundant voter is replicated, such as the various bus interfaces.

Now the reliability parameters for each case will be derived.

Reliability Parameters, Nonredundant System. The nonredundant system has a failure rate found by simple summation of the individual failure rates:

$$R_{non} = e^{-(Lp + 7Lm) * T}$$

Substituting the failure rates from Table 20 yields:

$$R_{non} = e^{-174.9 * T}$$

Lp	= 12.902	LSI-11 processor module.
Lm	= 23.139	Memory module (4K semiconductor RAM)

For the nonredundant voter design:

Lvp	= 1.963	Portion of voter connected to processor bus
Lvm	= 2.130	Portion of voter connnected to external bus
Lv	= 4.024	Portion of voter connected to both buses
Ln	= 11.708	Non-triplicated portion of voter

For the redundant voter design:

Lvp	= 3.311	Portion of voter connected to processor bus
Lvm	= 3.189	Portion of voter connnected to external bus
Lv	= 9.641	Portion of voter connected to both buses
Ln	= 3.165	Non-triplicated portion of voter (part of the control)

Table 20. Failure rates for system modules for different voter designs

Reliability Parameters, Triplicated System, Nonredundant Voter. The reliability of the triplicated system is found by summing the reliabilities for all states in which the system is correctly operating. Each state is a combination of working and failed units.

The various parts are:

Rp = reliability of processor bus elements
$= e^{-(Lp + Lvp) * T}$
$= e^{-14.865 * T}$

Rm = reliability of a single 4K memory module
$= e^{-Lm * T}$
$= e^{-23.139 * T}$

Rv = reliability of the triplicated part of the voter
$= e^{-Lv * T}$
$= e^{-4.024 * T}$

Rn = reliability of the non-triplicated part of the voter
$= e^{-Ln * T}$
$= e^{-11.708 * T}$

Re = reliability of the part of the voter associated with the external bus
$= e^{-Lvm * T}$
$= e^{-2.130 * T}$

Reliability Parameters, Triplicated System, Redundant Voter. If the voter is itself redundant, the reliability factors calculated above become:

$Rp = e^{-16.213 * T}$
$Rm = e^{-23.139 * T}$
$Rv = e^{-9.641 * T}$
$Rn = e^{-3.165 * T}$
$Re = e^{-3.189 * T}$

Reliability Model. The reliabilities for each operational state are:

1) At most one processor failed, at most one memory module per 4K address range failed, voter and buses all working.

$$R1 = (3Rp^2 - 2Rp^3) * (3Rm^2 - 2Rm^3)^7 * Rv^3 * Rn * Re^3$$

2) At most one processor failed, single memory bus failed, voter and all memory on the other two buses working.

$$R2 = 3 * (3Rp^2 - 2Rp^3) * Rm^{14} * Rv^3 * Rn * Re^2(1 - Re)$$

3) One third of voter failed, all processors and memories on the other two buses working.

$$R3 = 3 * Rp^3 * Rm^{14} * Rv^2(1 - Rv) * Rn * Re^2$$

The coefficient of three in R2 and R3 represents the three possible configurations for this error.

Note that in the last case, the failure of a third of the voter masks the operation of one processor-memory pair. Since it no longer matters whether that pair operates, the Rp and Re terms are only squared. When added together, the triplicated reliability reduces to:

$$R_{trip} = (3*Rm^2 - 2Rm^3)^7 * Rv^3 * Re^3 * Rn * (3Rp^2 - 2Rp^3) + 3 * (Rm^7 * Rv * Re * Rp)^2 * Rn * (Rv(2 - 3Re - 2Rp + 2Re*Rp) + 1)$$

These three reliabilities, R_{non} and the two cases of R_{trip}, are plotted using the derived reliability parameters in Figure 22. Note that the C.vmp system with a nonredundant voter is more reliable than the C.vmp system with a fully redundant voter except for reliabilities above 0.94, and a time period of less than 1600 hours.

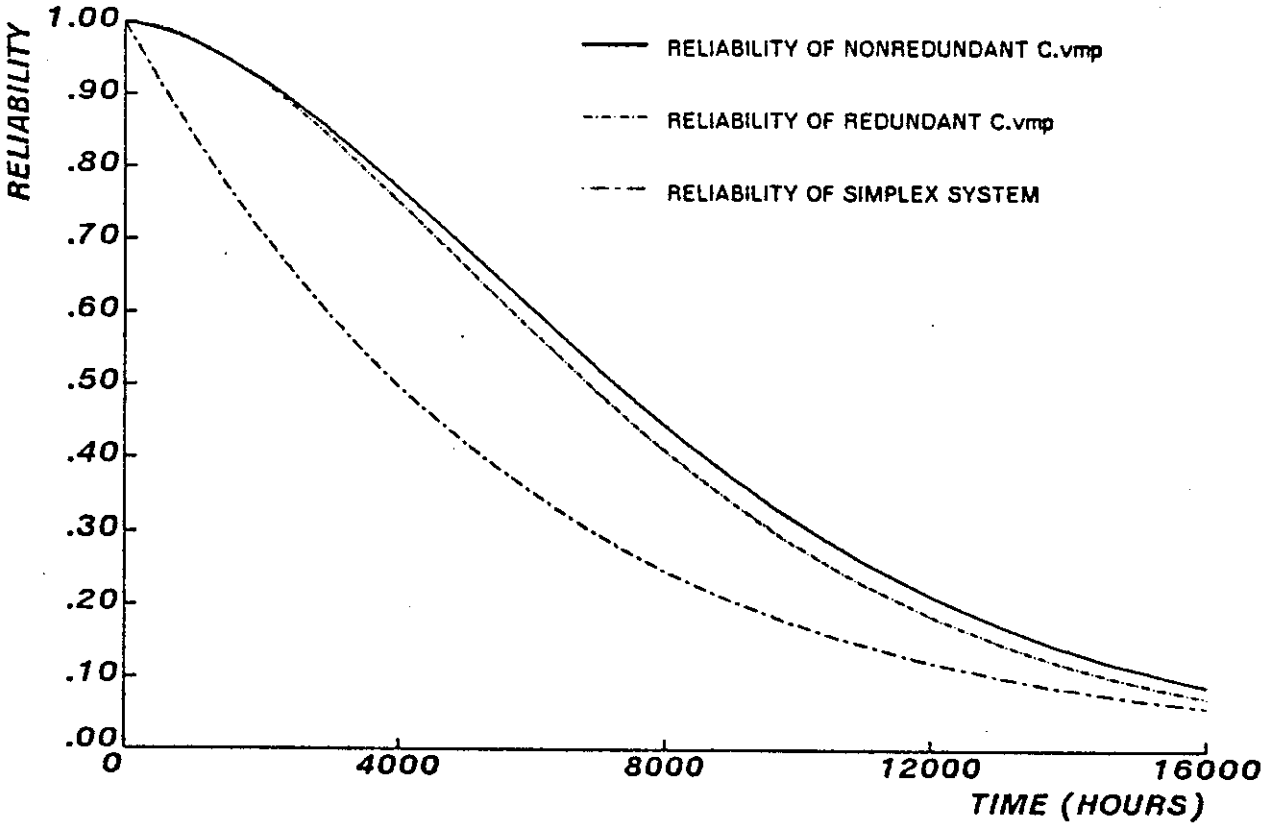


Figure 22. Reliabilities of Simplex system and C.vmp

4. Summary and conclusions

A modified MIL 217B model was proposed after comparing semiconductor chip vendor data to data from C-MU's multiprocessor systems. A program, AUTOFAIL, embodies a parameterized version of the modified model. AUTOFAIL was used to calculate failure rates for the various modules in the three multiprocessor systems. Hard failure reliability models were then presented using these calculated failure rates to provide a basis for consistent comparison.

After careful studies of the curves presented in the previous three sections, several interesting points are observed.

1- The lumped switch model for C.mmp shows the effect of dependency on a critical resource with relatively high failure rate. In Figure 9 (lumped switch) the curves for 4, 8, 12 required processors drops almost linearly with time cancelling the reliability improvement due to redundant processors and memory modules. Whereas in Figure 11 (distributed switch) the curves for 4, 8, 12 required processors are almost flat up to 2000 hours, pointing out the reliability improvement due to redundant modules (processors and memory modules). Substantial gain in reliability prediction can be obtained simply by more accurate modeling of the systems and their failure modes.

2- Requirements for a large number of one type of module will dominate the reliability improvement due to redundancy in other modules. In effect, the module with the least amount of redundancy behaves almost as a nonredundant system exhibiting a close to exponential failure rate. Figure 10 shows that requiring 75% of the memory modules negates the effect of redundant processors and all the curves for 4, 8, 12 required processors coincide with each other. This should be true for any architecture. Figure 17 illustrates the point for Cm*.

3- Small amounts of redundancy will improve system reliability. Beyond that limit, the addition of any more redundant modules does not incease system reliability. Figure 11 shows that for C.mmp, 8 redundant processors are the limit.

4- High reliability components will increase the reliability of the system dramatically.

5- The level of modularization can have an impact. A study not included in this paper examined the effect of 4Kw and 16Kw memory modules on Cm*. Due to the sharing of control circuitry, the failure rate of the 16Kw memory module was less than four times that of the 4Kw memory module. In cases where a low percentage of the total system memory was required, the 16Kw memory module was a better choice due to its lower failure rate. However, if a high percentage of the total system memory was required, the 4Kw memory module was superior since its lower level of modularity allowed a larger number of operational system states.

The three architectures presented here had differing design goals. C.mmp was meant to be a high performance multi-minicomputer. Cm* was designed to be a highly available, extensible, and modular mutiprocessor. C.vmp was a fault tolerant mutiprocessor with very limited expandability.

It is difficult to compare the three architecture in terms of reliablity. To get a feel for the absolute reliablity of each architecture, a typical curve for each one of the three, based on the actual hardware implementation, is plotted in Figure 23 The curve for C.mmp required 4 of 16 processors, 512Kw out of 1 Mw of memory and a distributed switch. Cm* required 3 of 9 processors and 144Kw out of 288Kw of memory. It is important to note that the architectures modelled in Figure 23 do not have the same processing power, however, C.mmp

appears best for short missions and C.vmp for long missions.

Perhaps the best way to compare these architectures is to normalize them with respect to a common factor, yielding differences from the actual existing implementations. This normalization will indicate the effect of the processor/memory interconnection structures on the system reliability. For example one of the common factors to normalize is the number of processors and memory modules. Another factor is performance. Studies are in progress at Carnegie-Mellon University to compare different interconnection structures under various normalizations.

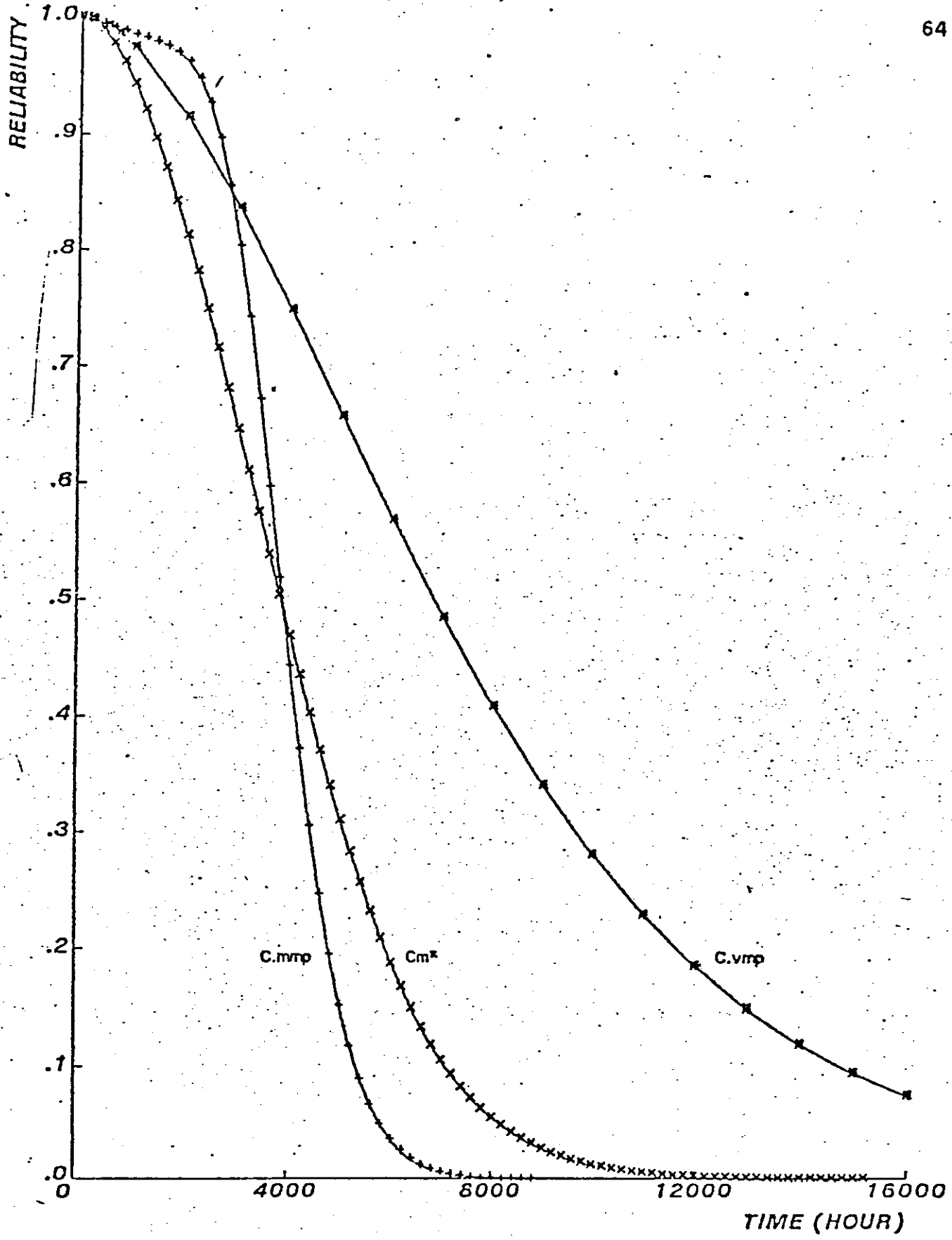


Figure 23. Comparison of reliabilities of C.mmp, Cm*, and C.vmp

5. Acknowledgements

We would like to thank Steve McConnel who derived the hard failure reliability model for C.vmp in its present form and Mickey Tsao who patiently provided valuable assistance in the preparation of the manuscript. Our gratitude also goes to the staff of the Engineering Lab in the Computer Science department at C-MU who painstakingly maintained the error logs for Cm* from which we extracted data used in Section 2.

REFERENCES

[1] D.P.Siewiorek, V.Kini, H.Mashburn, S.McConnel, M.Tsao, "A Case Study of C.mmp, Cm* and C.vmp -- I. Experiences with Fault Tolerance in Multiprocessor Systems", this issue.

[2] Mil-Std-Hdbk-217B, "Military Standardization Handbook: Reliability Prediction of Electronic Equipment", September 1974.

[3] H. Bellis, "Comparing Analytical Reliability Models to Hard and Transient Failure Data", M.S. Thesis, Department of Electrical Engineering, Carnegie-Mellon University, March 1978.

[4] Sturges, "The Choice of a Class Interval", Journal of the American Statistical Association, Vol. 21, pp. 65-66, 1926.

[5] H. Bellis, "Autofail - Automatic Failure Rate Calculator User Manual", Technical Report, Department of Electrical Engineering, Carnegie-Mellon University, May 1978.

[6] Algirdas Avizienis, "Architecture of fault-tolerant computing systems", Proc. international symposium on fault-tolerant computing, 1975.

[7] H.Wylie, G.J.Burnett, "Some relationships between failure detection probability and computer system reliability", AFIPS Conf. Proc., Fall Joint Comp. Conf. 1967, V.31, pp.745-756.

[8] W. G. Bouricius, W. C. Carter and P. R. Schneider "Reliability modeling techniques for self-repairing computer systems" Proc. 24th national conference ACM, 1969, page 295-309.

[9] Daniel P. Siewiorek, "Multiprocessors: Reliability Modeling and graceful degradation", Infotech state of art conference on system reliability, London 30 May- 1 June 1977, page 48-73

[10] Knox-Seith, J. K., "Improving the reliability of digital systems by redundancy and restoring organs", Solid-State Electronics Lab., Technical Report No. 4186-2, Stanford University, California, August 1964.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER CMU-CS-78-143	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) RELIABILITY IN MULTIPROCESSOR SYSTEMS: A CASE STUDY OF C.MMP, CM* AND C.VMP.		5. TYPE OF REPORT & PERIOD COVERED Interim 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Daniel Siewiorek, Vittal Kini		8. CONTRACT OR GRANT NUMBER(s) — NR-048-645
9. PERFORMING ORGANIZATION NAME AND ADDRESS Office of Naval Research Arlington, VA 22217		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Carnegie-Mellon University Computer Science Dept Pittsburgh, PA 15213		12. REPORT DATE 13. NUMBER OF PAGES 142
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Same as above		15. SECURITY CLASS. (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for Public Release; Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report consists of two papers that treat reliability of the multiprocessor systems at CMU. The first paper discusses the multiprocessor architectures, reliability features (hardware and software), and measured reliability data. The second paper presents hard failure data from one of the systems, calibrates a hard failure rate model, and analytically models the reliability of the three systems. These papers will appear in the October 1978 issue of the Proceedings of the IEEE (Special Issue on Fault Tolerant Systems).

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)