EMPHASIZING FAULT-TOLERANCE IN PARALLEL
COMPUTER SYSTEMS ARCHITECTURE

Bretholz Erich
Segal Zary
Strasbourger Edward

Technical Report No. 46
May 1975
ABSTRACT

Recent advances in the technology have made the design of inexpensive parallel fault tolerant computer systems possible. The present trend in the field is to provide systems with maximum availability and efficiency, to be used not only in time critical applications but also in general-purpose applications.

A survey is presented pointing out fault tolerancy in parallel computer systems. Duplex and reconfigurable computer systems are investigated.

The survey conclusion leads us to propose a reconfigurable, parallel, fault-tolerant computer architecture. The proposed system will perform reconfiguration upon job requirement and fault detection.

Some research directions on the proposed system are outlined and part of the problems are being investigated in the Technion Computer Science Department.

{Further details of the proposed system will be presented in another paper}
INTRODUCTION

Every parallel processor possesses an implicit degree of fault tolerance compared to a uniprocessor. Having several identical units and various interconnection paths between all units provides a higher redundancy in the parallel processor.

However, in designing a Parallel Computer System special attention should be paid to increasing the systems reliability and availability by making it Fault-Tolerant.

It is understood, in this paper, that availability is a measure of the probability that the system will perform its functions according to specifications. However, the availability is greatly affected by the recovery speed of a malfunctioning system and the reliability of its components.

By definition a Fault-Tolerant System is a computer system which overcomes the effects of failures in either units or interconnections between units up to a given extent.

Two approaches to this problem have been developed:
- reconfigurable computer systems ('fail-soft').
- duplex computer systems ('fail-safe').

Upon a fault detection in the Reconfigurable Computer System, the faulty unit or interconnection is located and isolated from the system and then the whole work-
load of the system is redistributed within the remaining system. Therefore the performance or throughput capability of the system degrades gradually.

In the Duplex System at least two units or complete systems work in parallel on the same problem and data. The intermediate and final results are compared. Nonmatching results indicates that at this level a malfunction has occurred. The malfunctioning unit is located and removed from the system. Although the total throughput capability remains unchanged, the fault detection ability of the remaining system decreases.

Although the duplex approach maintains the throughput capability upon a fault occurrence, it is wasteful approach because the redundant units do not improve the computation power of the whole system.

The Reconfigurable Computer System does not have this handicap.
DUPLEX COMPUTER SYSTEMS

As mentioned before, a duplex computer system is a computer system which possesses two or more identical functional units of each kind interconnected in such a way as to perform the same program on the same data at least twice at the same time. In normal operation the common result agrees with the personal result of each active processor. If disagreement occurs then a decision will be taken and a processor will be removed from operation.

![Diagram of Duplex Computer Systems - General Concept](image)

Fig. 1 - Duplex Computer Systems - General Concept

We would like to classify further this class of parallel computer systems according to the number and nature of the functional units incorporated in the system as follows:
1. Two processors working on the same problem e.g. Massachusetts Institute of Technology, SIRU system. [5]

2. Three processors working on the same problem e.g. NASA, Hamilton Standard Modular Computer. [5]

3. Several computers on-line with minimum of spares e.g. JET Propulsion Laboratory STAR. [18]

1. The basic problem in dual processor duplex system for accomplishing fault-tolerance seems to be the isolation of the faulty unit after a fault has been detected. A classical example in this area is the SAGE Air Defence System. One processor is an "active" one and the other "standby". The active machine transmits periodically a subset of intermediate results to the standby through a drum accessible to both processors. If this subset does not match the results obtained by the "standby", an error has been detected. To isolate the faulty processor a test program is executed. After switchover the new "active" would proceed from the last checkpoint where the intermediate results still matched.

Recently this concept was improved and a machine was developed at the Instrumentation Laboratory of MIT called SIRU (Fig. 2). To achieve inflight reliability and maintainability, hardware was supplied to perform fault detection, isolation and system reconfiguration.

Both processors are working on the same problem, executing the same instruction on the same data-stream in synchronisation. The "active" processor can
write on both memories, while each processor may read from it's assigned memory. The system is designed to perform continous checking of all units. The interconnection between all units permits an immediate reconfiguration, to establish the new active processor and its associated unit.

It can be seen from the above description that the designers of the system were mainly interested in maintaining high availability with minimum time loss.

Fig. 2 - SIRU / MIT Computer
2. Detection and isolation of the faulty unit of a triple processor system is done using the "voter" method also called Triple Modular Redundancy (TMR). By inserting a majority gate at the output of each set of replicated modules a fairly good estimation of the functioning of the modules can be gained.

For example consider a multiple input single output T.M.R. (Fig. 3). If $R_m$ is the reliability function of a module, $1 - R_m$ will be the probability that the module should fail. Assuming a system failure if two or more modules fail, this subsystem has a reliability of

$$R = R_m^2 + 3R_m^2 (1 - R_m).$$

However, the above result should be altered according to the logical function performed by the module. For example if in our case two modules have failed, then according to the above function the whole subsystem fails. Let us take a closer look at all the possible failure cases (table 1).

<table>
<thead>
<tr>
<th>Module 1</th>
<th>Module 2</th>
<th>Module 3</th>
<th>Voter</th>
</tr>
</thead>
<tbody>
<tr>
<td>Of</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>Of</td>
<td>Of</td>
<td>X</td>
<td>Of</td>
</tr>
<tr>
<td>Of</td>
<td>If</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>If</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>If</td>
<td>Of</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>If</td>
<td>If</td>
<td>X</td>
<td>If</td>
</tr>
<tr>
<td>X</td>
<td>Of</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

Table 1 - "Voter" fault propagation
We can see that in the indicated "system failure" cases (two out of three units failed) the failure does not propagate through the voter. Therefore the real estimated reliability function is

\[ R' = R_m^3 + 3R_m^2(1 - R_m) + K(3R_m)(1 - R_m)^2 \]

where \( K = \frac{\text{# of compensating failures}}{\text{# of any failures}} \)

If \( K = 0.5 \) then it can be shown that the reliability of the system has improved by 50%. It seems to us that using the former reliability estimation function leads to a theoretical availability which is lower by a significant percentage than the actual availability. One of the TMR systems developed by NASA at the Electronics Research Center is the modular Hamilton Standard Computer[5] (Fig. 4).
The system is composed of three identical sets each of which contains a Memory Unit, Control Unit, Arithmetic Unit and an Input/Output Unit. A Configuration Assignment Unit, implemented based on the "Voter" principle, interconnects functional units in order to maintain an operational system.

Fig. 4 - HAMILTON STANDARD Modular Computer
3. A many processor duplex system consists of a number of active subsystems which work together on a problem. Each active subsystem has some active units backed up by a set of standby spare units. A monitor subsystem continuously examines the active subsystems which work on the problem. Upon a fault detection the monitor initiates the detection of the faulty unit and its replacement by one of the spares. If a transient malfunction occurs, the detection and recovery mechanism performs its function at the first occurrence of damaged information leaving the faulty unit.

In order not to "overdesign" this type of system it is of utmost importance to establish the number of active units and of spare units to assure the problem solving requirements in a definite time interval.

Finding the minimum number of active and spare units suitable for a given problem solving in a given environment is known as the "coverage" of the problem. In other words it is the conditional probability that after a failure in the operation subsystem, the system is able to recover and continue information processing with no permanent loss of essential information.

Thus:

\[ c = p \left[ \text{System Recovers} \mid \text{Submodule Fails} \right] = A \cdot R \]

\[ A = p \left[ \text{Existence of adequate algorithm for detection and Recovery of the faulty module} \mid \text{Submodule Fails} \right] \]

\[ R = p \left[ \text{The Monitor has not failed} \mid \text{Submodule Fails} \right] \]
Given a single subsystem module set with N active and S spares and assuming exponential failure distribution with λ and μ failure rates of active and standby modules, a time dependant reliability recursive equation for one subsystem will be

$$R(N, S)[t] = e^{-N\lambda t} e^{-S\mu t} + (N\lambda + S\mu) \int_0^t e^{-(N\lambda + S\mu)x} R(N, S-1)[t-x]dx$$

The results of the above recursive equation (table 2) show the interdependance between choosing the numbers of spares relative to the number of active units and the reliability of the system.
\begin{table}
\centering
\begin{tabular}{|c|c|c|c|c|c|}
\hline
$\lambda t$ & 1/N & 2/N & 3/N & \ldots & N/N \\
\hline
0 & 1 & 1 & 1 & \ldots & 1 \\
0.05 & 0.9048 & 0.9931 & 0.9996 & \ldots & 1 \\
0.1 & 0.8187 & 0.9746 & 0.9968 & \ldots & 0.9996 \\
0.15 & 0.7048 & 0.9476 & 0.9903 & \ldots & 0.9983 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\
\hline
\end{tabular}
\caption{Table 2}
\end{table}

To choose the number of spares in such a system given the number of active units and the availability requirement, use should be made of the above results. A system based on the above theoretical approach has been developed at the Jet Propulsion Laboratory named JPL STAR (Self Testing And Repairing) [18] (Fig. 6)
The system is built around a processor called Test and Repair Processor which re-starts the program from a checkpoint if an error was detected. The error detection is performed as follows: each 32 bit word has four check bits. The check bits are the remainder of N\#15.

If an error is detected it is considered a transient error and a program restart from the previous checkpoint is undertaken. If the error occurs twice consecutively in one module it is considered a permanent error. The unit is replaced by a standby spare and the program restarted from the previous checkpoint.

**RECONFIGURABLE COMPUTER SYSTEMS**

In time dependent applications such as military or space problems it is of utmost importance to maintain a given throughput capability of the computer system. Therefore usually one of the solutions outlined previously is used.

It is obvious that just a small part of the system capacity is used and therefore the whole equipment efficiency is low.

In other classes of application such as time sharing, batch production systems, distributed computing, and computer networks, where a high equipment efficiency is expected in order to cut prices, the fail-soft approach is attractive. The great cost advantage of this technique is provided from the fact that no redundant idle backup units are needed. The architecture of a fail-soft system must allow the reconfiguration of resources so that the failure of one unit does not cause other units associated with it to be unusable.
Therefore in order that a nonredundant system to be fail-soft it must fulfil the following requirements:

1. Modular architecture with multiple functional units of each type.
2. Failure detection and identification
3. Isolation of the faulty unit
4. System reconfiguration in order to continue the computation (although with a lower throughput).

From the above requirements it seems to be clear that a multi-processor system is needed in order to utilise the maximum capability of all units of the system.

However in order to attain a high degree of fault tolerance the system must be able to redistribute its resources. The memory should be modular and switchable between all processors. All I/O units should be able to attach to any process. A special bus architecture is needed to provide flexible interconnection. In a multiprocessor environment one of the three interconnection schemes is used:

- time shared unibus
- crossbar switch matrix,
- multibus.

Comparing these methods according to their reliability, the unibus approach is the weakest because a physical damage of the bus will cause the loss of the whole system. In the crossbar matrix the reliability of the system depends mainly on the reliability of the switch. If it fails it causes the loss of one interconnection path between two functional units.

To increase the reliability of the unibus method a hybrid bussing method has been developed. It uses a permanent global bus between all units and local
switchable (under program control) busses between processors. If a failure is detected on the global bus an alternate path is built by switching the local bus.

In a multiprocessor environment where a multiplicity of processors are active concurrently, two types of violations can occur with respect to a given process:
- unauthorized interference by another process
- internal failures in the process without interference from another process.

Interprocess failures lead to contamination of the whole system while intraprocess failures have an influence only on the given process. The same methods are used for fault detection as described in the duplex system.

Repartitioning of the computer system into operational subsystems is a basic requirement for reliable operation. Control tables of the operating system must contain all operational resources in the system at a given time and the interconnection between them. After a fault detection and logic isolation the operating system, using specialized hardware, physically isolates the faulty units and reconfigures.
the remaining resources according to the process requirement and continues computation. The system performance is slightly degraded. However, the system is still able to satisfy the process requirement. Automatic reconfiguration and system recovery upon fault occurrence is performed in two ways:

- UNIVAC 1108 (fig. 8) uses a dedicated processor, the Availability Control Unit, which controls the interfaces to all units, thus providing means for physical isolation of elements and system reconfiguration. The weakness of this system is exactly the centralized control of isolation and reconfiguration of the whole system. A failure in this unit has catastrophic effect on the whole system.

![Diagram of UNIVAC 1108](Fig. 8 - UNIVAC 1108)
In IBM 9020, as opposed to the UNIVAC, a distributed control of isolation and reconfiguration was implemented.

Each functional unit contains a dedicated register called Configuration Control Register. Each CCR contains the unit status and its relationship to other elements in the system.

The task of dynamic reconfiguration can be performed by any processor (but only one) in control status executing a privileged instruction which accesses the CCR of the referred unit. Therefore a floating distributed control is achieved.

SOFTWARE GUIDELINES FOR A FAULT TOLERANT OPERATING SYSTEM

Additional attention should be paid, in designing the operating system of a Parallel Computer System, to provide the fault tolerance characteristics. We can divide the existing operating systems into three classes:

- Master slave
- Semi-independent operating subsystem residing in each processor
- Floating control

The master slave operating system is appropriate for a duplex computer system which uses one Monitor Unit to control the actions of the system. The main characteristics of this master slave operating system are:

- the supervisor runs in the Monitor Unit (only one at a time)
- most of the supervisor routines should not be reentrant because only the Monitor Unit is executing them.
This will protect against access conflict to the system assignment tables.

In order to achieve high availability, the operating system must be able to continue its operation if a fault is detected in the Monitor or Unit itself and is replaced by a backup unit. The main advantage of such a system is its relative simplicity, there being no need for the system to handle various conflict situations of the operating system itself. However this simplicity is paid for by a loss of flexibility.

In the semiindependent operating subsystem each processor executes its needed supervisor functions independently. Therefore the system is characterised by:

- all supervisor routines must be reentrant or reusable or an unaltered copy of each must be available to each processor.

- each supervisor will build and update its own private tables concerning the status of all processors and resources allocated to it.

- the operating subsystem should be able to handle I/O activities with the I/O equipment assigned to it.

It appears that a malfunction in one subsystem will not propagate to the others. Therefore this system is not subject to catastrophic failures. If a failure has been detected, one of the other available subsystems must get control and perform tests of the failing subsystem in order to detect and isolate the faulty unit and return to the system the remaining operational units.

In order to perform the above function the third class of operating systems has been introduced.
The architecture of a system which contains a multiple number of similar units and several ways of interconnecting between them can be treated as a symmetric hardware configuration. Based on this, each processor can be built having the ability to act as monitor of the remaining system. Therefore, a floating control operating system promotes maximum flexibility and availability of the system. Examining a process performing in such an environment, it seems that there are two kinds of control services needed: overall system control functions and local system control functions. The overall system control functions are:

- job scheduling
- interprocess communication control
- system files management
- asynchronous I/O handling
- interprocess protection
- resource allocation
- test, initialisation, fault detection and isolation

The local control functions are:

- multiprogramming task management
- intraprocess task scheduling
- local files management
- private table management
- resource deallocation
- process errors and violation detection
- interprocess communication
- assigned I/O device handling
- local test functions
The advantages of this type of operating system consist first of all in better efficiency in resource usage, providing an even load balance. Having a floating control feature and a hardware symmetry, this system will provide graceful degradation and implicitly better availability. The floating control system properties are adequate for use in a reconfigurable fail-soft computer structure. Having a distributed part of control (local control) this system is able to implement different machines on different processors.

LOOKING TOWARD A RECONFIGURABLE PARALLEL PROCESSOR

The previous survey on fault-tolerant computer systems is intended to present the state of the art of the work which has been done in the field.

As a conclusion to the previous survey we assert that the existing systems either waste part of the existing resources (in duplex systems) or provide reconfigurability only after fault occurrence (reconfigurable, fail-soft systems).

From the users point of view, the main interest is in performing some work in a given period of time. The work to be done is running programs on data. There are two kinds of programs relative to parallelism:
- with external parallelism; each program can run serially.
- with internal parallelism; parts of a single program can be performed in parallel.
To solve efficiently external parallelism a multiprogramming or/multiprocessor general purpose computer environment can be used. To solve the intrinsic parallelism efficiently a special purpose computer must be used (e.g. pipeline processor array processor etc.)
The data, the program can act upon, can come from different sources: files stored on magnetic devices, data provided by users at terminals, data provided by the external physical world via sensors, or data provided by other computer systems.

The proposed system will provide a flexible and fault tolerant environment efficiently using the available resources to solve all outstanding problems. The system should be easily user tailored in the sense that the user should be able to get the minimum workable configuration which suits his application. The system could therefore, be used as a general purpose computer, as a scientific computer, as an industrial control system, or in military application.

In designing the system we have to consider that the system should:

1. Gracefully degrade upon fault occurrences.
2. Have modular functional units in order to permit expandability and adaptability.
3. Be reconfigurable.
4. Provide an easy way of connecting resources.
5. Support the requirement for all software to be able to execute on subsystems of the proposed system.
6. Provide a reasonable amount of resources and suitable mechanisms for virtual storage implementation.
7. Provide interprocess integrity

To achieve the above goals we took into consideration:
- the present state of the art in the LSI world; such as high-speed, inexpensive, flexible microporcessors, with inteligent I/O, integrated PROM, ROM, RAM, progr-

ramable.
crosspoint switches, ALU, special integrated execution units (e.g. pipeline, stack, FFT, correlator) inexpensive I/O devices (e.g. floppy disk, tape-cassette).

- Microprogramming seems to be the most efficient way to perform emulation, bootstrap loading of operating system, special control function.
- Fast I/O (e.g. DMA) will provide the virtual storage mechanism
- To assure interprocess integrity each subsystem will have its own control.
- Only one of the processors at a time can be in control of the entire system (a Supervisor Processor) but any of the remaining processors can take over the Supervisor function as required. Therefore the system has a floating control.
- To efficiently use resources a multiprogramming environment should be provided in the subsystems.
- Fault detection, fault isolation, and system monitoring are provided.
- Communication between subsystem will be performed through SP. (Supervisor Processor).
Fig. 9 - BLOCK DIAGRAM

MM - Memory Module
MI - Memory Interface
IPU - Instruction Processor Unit
RC - Reconfiguration Control
DCR - Distributed Control Register
RR - Reconfiguration Register
I/O - Reconfigurable Bus
M/P - Reconfigurable Bus
IPU/EXU - Reconfigurable Bus
EXU - Execution Unit
PRINCIPLE OF OPERATION

We define a subsystem as a portion of the system able to perform a given job. Logically a subsystem will contain all functional units, (see fig. 10)

Fig. 10 - SUBSYSTEM K

At any given time one of the subsystems will control the entire system, performing all supervisor and monitoring functions. This subsystem will be called the Supervisor Processor (SP). All other subsystems will be in a problem solving state, and therefore will be called Problem Processors (PP). Any subsystem can be in one of the following states:

- supervisor state
- fault state
- problem state
- under test
- unused state
- presumptive error condition
Only one subsystem at any time will control the entire system, performing all supervisor and monitoring functions (SP). Some of the SP tasks are:

- to maintain the system or part of it in good working condition
  - initiate tests.
  - perform fault detection & isolation.
  - perform system reconfiguration after fault occurrence.

- job scheduling
  - schedule the jobs to existing subsystems if the resources allocated to it correspond to the job requirement.

  if no such subsystem is presently configured the job is entered on the corresponding class queue waiting for the subsystem class to be configured.

- resource reconfiguration according to class requirement
  - the class is determined by memory, processing units, execution units and auxiliary storage requirements.

- interprocess communication
  - provide interprocess interlock.
  - poll all stations for outstanding service request.

  resolve special I/O functions not allocated to subsystems (i.e. printer, punch, tapes, card reader etc.).

The PP provides the wanted environment for the job class execution. Therefore, it represents a self-contained system. It performs all needed supervisory functions as:

- I/O for the allocated resources.
- Program interruption traps.
- Self-testing.
- SP testing.
- multi-tasking services.
- communication with SP or other PP's via SP.
1. After enabling the system a subsystem is chosen automatically and assigned as SP. By manual operator intervention the automatic choice of SP can be overriden.

2. A TEST of all subsystems is performed by SP. All units which fail the test are marked.

3. Jobs are read from various I/O devices connected to SP and are enqueued on the JOBQUEUE according to requirement classes.

4. When one of the classes has enough work to be started, a subsystem is configured, if all requested units are available.

5. The job class is assigned to the new created subsystem.

6. The PP Supervisor initiates the allocated jobs, according to priority, assists them in all I/O requests, and terminates the jobs. After completion of each job SP is informed.

7. A TEST is performed by SP on the subsystem. If TEST succeeds and there are more jobs in class then proceed to 6. If no more jobs in class, the subsystem units are marked as available.

8. If TEST fails, both subsystems SP and PP are marked with Presumptive Error Check (PEC).

9. The next available PP will become SP and performs TEST on all subsystems marked PEC. If TEST fails on all tested subsystems the SP will be marked PEC and proceed to 9.

10. If TEST succeeds on all tested subsystems an intermittent error is marked and all PEC's are reset. If the number of intermittent errors exceeds a given number, all units are marked as failing.
PEC - Presumptive ERROR Check

tot. faulty = \sum PEC's

Fig. 11 - Test Procedure

Perform TEST on PP

TEST failed

NO

RETURN

YES

MARK SP & PP as PEC

Any subsystem free?

no

YES

Free Subsystem loads SP functions; \# faulty = 0

TEST processor with PEC

TEST O.K

NO

\# faulty = \# faulty + 1

\# faulty >\thetat

\# faulty > \# faulty

yes

no

\# faulty > 0

no

Mark processor with outstanding PEC as faulty, inform monitor

Mark PEC on this PP

\# faulty > 0

yes

no

increase intermittent error count of processors, inform monitor

Reset all PEC

intermittent > maxinterm

yes

mark processor as faulty

RETURN

no

Reset all PEC

no

Mark processor with outstanding PEC as faulty, inform monitor

Mark PEC on this PP

\# faulty > 0

yes

no

increase intermittent error count of processors, inform monitor

Reset all PEC

intermittent > maxinterm

yes

mark processor as faulty

RETURN

no

Mark processor with outstanding PEC as faulty, inform monitor

Mark PEC on this PP

\# faulty > 0

yes

no

increase intermittent error count of processors, inform monitor

Reset all PEC

intermittent > maxinterm

yes

mark processor as faulty

RETURN

no

Mark processor with outstanding PEC as faulty, inform monitor

Mark PEC on this PP

\# faulty > 0

yes

no

increase intermittent error count of processors, inform monitor

Reset all PEC

intermittent > maxinterm

yes

mark processor as faulty

RETURN

no

Mark processor with outstanding PEC as faulty, inform monitor

Mark PEC on this PP

\# faulty > 0

yes

no

increase intermittent error count of processors, inform monitor

Reset all PEC

intermittent > maxinterm

yes

mark processor as faulty

RETURN

no

Mark processor with outstanding PEC as faulty, inform monitor

Mark PEC on this PP

\# faulty > 0

yes

no

increase intermittent error count of processors, inform monitor

Reset all PEC

intermittent > maxinterm

yes

mark processor as faulty

RETURN

no

Mark processor with outstanding PEC as faulty, inform monitor

Mark PEC on this PP

\# faulty > 0

yes

no

increase intermittent error count of processors, inform monitor

Reset all PEC

intermittent > maxinterm

yes

mark processor as faulty

RETURN

no

Mark processor with outstanding PEC as faulty, inform monitor

Mark PEC on this PP

\# faulty > 0

yes

no

increase intermittent error count of processors, inform monitor

Reset all PEC

intermittent > maxinterm

yes

mark processor as faulty

RETURN

no

Mark processor with outstanding PEC as faulty, inform monitor

Mark PEC on this PP

\# faulty > 0

yes

no

increase intermittent error count of processors, inform monitor

Reset all PEC

intermittent > maxinterm

yes

mark processor as faulty

RETURN

no

Mark processor with outstanding PEC as faulty, inform monitor

Mark PEC on this PP

\# faulty > 0

yes

no

increase intermittent error count of processors, inform monitor

Reset all PEC

intermittent > maxinterm

yes

mark processor as faulty

RETURN

no

Mark processor with outstanding PEC as faulty, inform monitor

Mark PEC on this PP

\# faulty > 0

yes

no

increase intermittent error count of processors, inform monitor

Reset all PEC

intermittent > maxinterm

yes

mark processor as faulty

RETURN

no

Mark processor with outstanding PEC as faulty, inform monitor

Mark PEC on this PP

\# faulty > 0

yes

no

increase intermittent error count of processors, inform monitor

Reset all PEC

intermittent > maxinterm

yes

mark processor as faulty

RETURN

no

Mark processor with outstanding PEC as faulty, inform monitor

Mark PEC on this PP

\# faulty > 0

yes

no

increase intermittent error count of processors, inform monitor

Reset all PEC

intermittent > maxinterm

yes

mark processor as faulty

RETURN

no

Mark processor with outstanding PEC as faulty, inform monitor

Mark PEC on this PP

\# faulty > 0

yes

no

increase intermittent error count of processors, inform monitor

Reset all PEC

intermittent > maxinterm

yes

mark processor as faulty

RETURN

no

Mark processor with outstanding PEC as faulty, inform monitor

Mark PEC on this PP

\# faulty > 0

yes

no

increase intermittent error count of processors, inform monitor

Reset all PEC

intermittent > maxinterm

yes

mark processor as faulty

RETURN

no

Mark processor with outstanding PEC as faulty, inform monitor

Mark PEC on this PP

\# faulty > 0

yes

no

increase intermittent error count of processors, inform monitor

Reset all PEC

intermittent > maxinterm

yes

mark processor as faulty

RETURN

no

Mark processor with outstanding PEC as faulty, inform monitor

Mark PEC on this PP

\# faulty > 0

yes

no

increase intermittent error count of processors, inform monitor

Reset all PEC

intermittent > maxinterm

yes

mark processor as faulty

RETURN

no

Mark processor with outstanding PEC as faulty, inform monitor

Mark PEC on this PP

\# faulty > 0

yes

no

increase intermittent error count of processors, inform monitor

Reset all PEC

intermittent > maxinterm

yes

mark processor as faulty

RETURN

no

Mark processor with outstanding PEC as faulty, inform monitor

Mark PEC on this PP

\# faulty > 0

yes

no

increase intermittent error count of processors, inform monitor

Reset all PEC

intermittent > maxinterm

yes

mark processor as faulty

RETURN

no

Mark processor with outstanding PEC as faulty, inform monitor

Mark PEC on this PP

\# faulty > 0

yes

no

increase intermittent error count of processors, inform monitor

Reset all PEC

intermittent > maxinterm

yes

mark processor as faulty

RETURN

no

Mark processor with outstanding PEC as faulty, inform monitor

Mark PEC on this PP

\# faulty > 0

yes

no

increase intermittent error count of processors, inform monitor

Reset all PEC

intermittent > maxinterm

yes

mark processor as faulty

RETURN

no

Mark processor with outstanding PEC as faulty, inform monitor

Mark PEC on this PP

\# faulty > 0

yes

no

increase intermittent error count of processors, inform monitor

Reset all PEC

intermittent > maxinterm

yes

mark processor as faulty

RETURN

no

Mark processor with outstanding PEC as faulty, inform monitor

Mark PEC on this PP

\# faulty > 0

yes

no

increase intermittent error count of processors, inform monitor

Reset all PEC

intermittent > maxinterm

yes

mark processor as faulty

RETURN

no

Mark processor with outstanding PEC as faulty, inform monitor

Mark PEC on this PP

\# faulty > 0

yes

no

increase intermittent error count of processors, inform monitor

Reset all PEC

intermittent > maxinterm

yes

mark processor as faulty

RETURN

no

Mark processor with outstanding PEC as faulty, inform monitor

Mark PEC on this PP

\# faulty > 0

yes

no

increase intermittent error count of processors, inform monitor

Reset all PEC

intermittent > maxinterm

yes

mark processor as faulty

RETURN

no

Mark processor with outstanding PEC as faulty, inform monitor

Mark PEC on this PP

\# faulty > 0

yes

no

increase intermittent error count of processors, inform monitor

Reset all PEC

intermittent > maxinterm

yes

mark processor as faulty

RETURN

no

Mark processor with outstanding PEC as faulty, inform monitor

Mark PEC on this PP

\# faulty > 0

yes

no

increase intermittent error count of processors, inform monitor

Reset all PEC

intermittent > maxinterm

yes

mark processor as faulty

RETURN

no

Mark processor with outstanding PEC as faulty, inform monitor

Mark PEC on this PP

\# faulty > 0

yes

no

increase intermittent error count of processors, inform monitor

Reset all PEC

intermittent > maxinterm

yes

mark processor as faulty

RETURN

no

Mark processor with outstanding PEC as faulty, inform monitor

Mark PEC on this PP

\# faulty > 0

yes

no

increase intermittent error count of processors, inform monitor

Reset all PEC

intermittent > maxinterm

yes

mark processor as faulty

RETURN

no

Mark processor with outstanding PEC as faulty, inform monitor

Mark PEC on this PP

\# faulty > 0

yes

no

increase intermittent error count of processors, inform monitor

Reset all PEC

intermittent > maxinterm

yes

mark processor as faulty

RETURN

no

Mark processor with outstanding PEC as faulty, inform monitor

Mark PEC on this PP

\# faulty > 0

yes

no

increase intermittent error count of processors, inform monitor

Reset all PEC

intermittent > maxinterm

yes

mark processor as faulty

RETURN

no

Mark processor with outstanding PEC as faulty, inform monitor

Mark PEC on this PP

\# faulty > 0

yes

no

increase intermittent error count of processors, inform monitor

Reset all PEC

intermittent > maxinterm

yes

mark processor as faulty

RETURN

no

Mark processor with outstanding PEC as faulty, inform monitor

Mark PEC on this PP

\# faulty > 0

yes

no

increase intermittent error count of processors, inform monitor

Reset all PEC

intermittent > maxinterm

yes

mark processor as faulty

RETURN

no

Mark processor with outstanding PEC as faulty, inform monitor

Mark PEC on this PP

\# faulty > 0

yes

no

increase intermittent error count of processors, inform monitor

Reset all PEC

intermittent > maxinterm

yes

mark processor as faulty

RETURN

no

Mark processor with outstanding PEC as faulty, inform monitor

Mark PEC on this PP

\# faulty > 0

yes

no

increase intermittent error count of processors, inform monitor

Reset all PEC

intermittent > maxinterm

yes

mark processor as faulty

RETURN

no

Mark processor with outstanding PEC as faulty, inform monitor

Mark PEC on this PP

\# fault
Fig. 12 - Job Flow
11. If at least one subsystem has failed the TEST and at least one has succeeded, then all units which failed the test will be marked "failed" and all other PEC's will be reset, the subsystem becoming available again.

12. All jobs which have been performed on all the faulty subsystems will be reentered on the JOBQUEUE.

Interprocess communication is performed by SP using polling and addressing techniques. SP will act mainly as an interlock for PP to PP communication while processing the messages which are addressed to it.

SAMPLE SUBSYSTEM CONFIGURATION

The Supervisor Processor SP will contain the minimum workable configuration: a memory module with a memory interface, an Instruction Processor Unit with its related I/O Controller, and an Arithmetic Logic Unit as its EXU.

SP controls the Reconfiguration Control BUS. Through the I/O Reconfigurable Bus SP is connected to the system's Disk & to the slow I/O units (printers, card readers, punches and tapes). SP is also connected to the I/O address which allows interprocess communication. SP polls the requests to this address.

The Program Processor (PP) will contain a requested number of memory modules with their interfaces, an Instruction Processor Unit with the related I/O Controller, and Execution Units. It will also have access to the I/O address for interprocess communication where the PP can ask for communication with another processor.
Let us explain the interconnection and function of the basic units analysing a Problem Processor subsystem. (See Fig. 13) The configuration is suitable for a job class requiring:

- \( m \) primary memory modules
- one Instruction Processing Unit
- three Execution Units
  - Arithmetic Logic Unit
  - Decimal Arithmetic Unit
  - Complex Arithmetic Unit
- \( n \) secondary memory units (floppy disk)
- \( P \) terminals concentrated by a front end processor

The Supervisor Processor has prepared the environment to permit the execution of the specific job class. The required resources are made available to the subsystem using a subsystem resource table. The bootstrap located in the ROM is started by SP and then SP ceases to control the subsystem until required by PP. The bootstrap loads the local supervisor. The local supervisor provides the environment and supervisor services required by the jobs.

Interprocess communication is provided via SP using the "mailbox" concept. A request for communication is raised in the Distributed Control Register. The SP polls the DCR's and if a request is found it is serviced, accessing the agreed "mailbox".
Fig. 13 - Sample Problem Subsystem Configuration
One DCR is related to each IPU and contains the state of the subsystem (SP state, PP state, program state, available state, etc.) All DCR's are connected by a bus. Each IPU has an associated Reconfiguration Control register which allows it to control the Reconfigurable Control Bus.

**THE PIPELINE CONFIGURATION**

This is one of the special PP configurations. In this configuration one of the IPU will be in control of the entire subsystem, decoding and executing the instructions. If a pipeline instruction is detected, commands will be given to all IPU's in the subsystem. According to the commands they will start fetching data from their attached memory modules and passing the data to the Pipeline Execution Unit. This approach has been chosen in order to increase the data rate to and from the Pipeline Execution Unit. Overlapping the memory will allow the pipeline instruction to be executed with high speed while the serial part of the program will be executed in the master IPU of this subsystem.

![Fig. 14 - Pipeline Subsystem](image)
RESEARCH PROBLEMS

It seems to us that in such a concept of a computer system little research has been done so far. Therefore it is a lucrative spot in the field.

Some of the outstanding research problems are:

- the efficiency of the distributed control mechanism.
- when is it desirable to run a job on an existing subsystem and when should the configuration be changed for the job class requirements.
- resource allocation algorithm
- test programs for variable configuration subsystems
- job class specifications.
- system efficiency evaluation:
  
  supervisor processor utilization
  reconfiguration overhead
- system reliability and availability evaluation
- the efficiency of the I/O mechanism
- analysis of system integrity

The above problems may be solved at three levels:

1. theoretical study
2. simulation
3. implementation
CONCLUSION

In this paper we have presented a short survey of the state of the art of fault tolerant computer systems, emphasizing the strong relationship between fault tolerant machines and parallel computer systems.

It has been outlined that the desired system should be a parallel, fault tolerant, fail soft computer system with floating control.

Based on the above conclusion, a computer system has been proposed. The system is built up around an intelligent reconfigurable crossbar switch bus. The implementation has become possible due to the availability of inexpensive LSI circuits, microprocessors, memory modules, and I/O units.

The proposed system maximizes resource utilization and system availability.
REFERENCES

A MULTIPURPOSE PROCESSOR ENHANCEMENT STRUCTURE - COMPCON 1972

A FAIL-SOFTLY SYSTEM FOR TIME-SHARING USE - IEEE FAULT TOLERANT COMPUTING 1972

AN ANALYSIS OF 'PRIME' USING A NEW RELIABILITY MODEL - IEEE FAULT TOLERANT COMPUTING 1974

DESIGN OF A MICROPROGRAM CONTROL FOR SELF-CHECKING - IEEE FAULT TOLERANT COMPUTING 1972

[5] Ph. E. ENSLOW
MULTIPROCESSORS AND PARALLEL PROCESSING - I. WILLEY & SONS 1974

FAULT TOLERANT PROGRAMMING - IEEE-FTC 1972

SHARED RESOURCE MULTIPROCESSING - COMPUTER 5 (1972)

CONTROL DATA STAR 100 PROCESSOR DESIGN - COMP-CON 1972

[9] HOBS (EDITOR)
PARALLEL PROCESSES, SYSTEMS, TECHNOLOGIES AND APPLICATIONS - SPARTAN BOOKS 1970

[10] M.P. HAMER
RELIABILITY MODELLING CONSIDERATION FOR REAL-TIME CONTROL SYSTEM - IEEE FTC-4-1974

THE ARCHITECTURAL ELEMENTS OF SYMMETRIC FAULT-TOLERANT MULTIPROCESSOR - IEEE - FTC-4-1974
[12] P.D. JONES
IMPLICIT STORAGE MANAGEMENT IN THE CONTROL DATA STAR-100 - COMPCON 1972

STAR - A DE-EDUCATION PROBLEM - COMCON 1972

[14] H. LORIN
PARALLELISM IN HARDWARE AND SOFTWARE: REAL AND APPARENT CONCURRENCY - PRENTICE HALL 1972

G.M.R: CENTRAL MODULAR REDUNDANCY - IEEE FTC-4 1974

[16] J.I. MASTRANADI
FAULT TOLERANT STORAGE SYSTEM DESIGN STUDIES - IEEE FTC-3 1973

[17] J.E. REQUA
STAR: A SYSTEM PROGRAMMER'S VIEW - COMCON 1972

[18] J.A. ROHR
STAREX SELF-REPAIR Routines: SOFTWARE RECOVERY IN THE JPL-STAR COMPUTER - IEEE FTC-3 1973

[19] D.A. RENNELS
RMS - A RELIABILITY MODELING SYSTEM FOR SELF REPAIRING COMPUTERS - IEEE FTC-3 1973

[20] D.P. SIEWIOREK
RELIABILITY MODELING OF COMPENSATING MODULE FAILURES IN MAJORITY VOTED REDUNDANCY - IEEE FTC-4 1974

[21] T.B. SMITH
A. DAMAGE and FAULT-TOLERANT INPUT/OUTPUT NETWORK - IEEE FTC-4 1974

[22] I.J. STIFFLER
THE SERF FAULT-TOLERANT COMPUTER - IEEE FTC-3 1973

[23] J.H. WENSLEY, K.N. LEVITT, p.g. neuman
A COMPARATIVE STUDY OF ARCHITECTURE FOR FAULT-TOLERANCE - IEEE FTC-4 1974