DESIGN OF FAULT-TOLERANT ASSOCIATIVE PROCESSORS*

Behrooz Parhami
Algirdas Avižienis
Computer Science Department
University of California, Los Angeles

ABSTRACT

Recent advances in computer technology have made the design of large and very flexible associative processors possible. Such systems are extremely complex and must be adequately protected against failures if they are to be used in critical application areas such as air traffic control or for performing control functions in fault-tolerant computers. This paper summarizes the results of a study which has indicated the techniques that are applicable in the design of fault-tolerant associative processors. Associative processors are divided into four classes of fully parallel, bit-serial, word-serial, and block-oriented systems. A technique for modularizing the design of an associative processor is given. The detection of errors within modules is discussed for the four classes mentioned above. Several schemes for reconfiguration are discussed which allow us to establish an appropriate intercommunication pattern after replacing the faulty module by a spare. The design of a fault-tolerant associative processor, which uses some of the techniques discussed previously, is presented.

BACKGROUND

Associative processors are of interest since they enable us to solve many data processing problems for which digital computers with conventional architectures are either unsuitable or highly inefficient. Based on the applications that have been proposed for associative processors, there are at least two reasons for studying the fault tolerance problems of such devices: (1) In some proposed application areas, such as air traffic control [1], the effect of an undetected fault-induced error may be catastrophic. (2) To be able to perform control functions [2] in a fault-tolerant computer, an associative device must itself be fault-tolerant, since, otherwise, it will become part of the system's hard core and will contribute heavily to its unreliability. In addition, the extreme complexity of large, general-purpose associative processors necessitates the incorporation of fault tolerance features into their design.

It is remarkable, therefore, that the problem of fault-tolerance of associative devices has remained virtually untouched. Ewing and Davies [3] give techniques for coping with some hardware malfunctions in a plated-wire implementation of a particular associative processor. Proudman [4] suggests that a single error correcting code can be used in conjunction with mismatch detectors with a threshold of 2 to detect storage errors. This paper summarizes the results of a study on fault tolerance techniques for associative processors [5]. We will concern ourselves with hardware faults and will assume the programs to be correct representations of intended algorithms for the specified domain of operation. We may note, however, that the simplified software of associative processors (e.g. fewer loops) with respect to conventional systems, results in a proportional simplification in the problem of software fault tolerance.

In the remainder of this paper, we will refer to fully parallel, bit-serial, word-serial, and block-oriented architectures for associative processors. This classification, which is based on the degree of parallelism in operations or, alternatively, the amount of storage associated with each unit of processing logic, is described briefly as follows. A more detailed discussion of these concepts and a comprehensive set of references can be found in [6].

(1) In fully parallel associative processors, processing logic is associated with each bit of stored data. Most fully parallel systems implement only the exact-match search operation in hardware and use software techniques for arithmetic, logic, and more complex searches.
(2) In bit-serial associative processors, processing logic is associated with each word of stored data. All the words can be processed in parallel, each in a bit-serial manner.
(3) In word-serial associative processors, a single processing unit operates serially on all the words. This approach essentially represents hardware implementation of a simple program loop which is used for linear search.
(4) In block-oriented associative processors, one block of information is associated with a unit of processing logic. A low-cost implementation of such a system may use a head-per-track magnetic recording memory in which each block is stored on one or more tracks.

FAULT TOLERANCE APPROACH

Figure 1 shows a model for an associative processor which applies to the three classes of fully parallel, bit-serial, and block-oriented systems. Since word-serial associative processors closely resemble conventional systems, their fault tolerance problems can be studied separately. Each processing element (PE) in

*This research was supported by the U.S. National Science Foundation under Grant No. GJ 33007X

Figure 1 consists of one unit of processing logic and its associated storage elements. In general, the processing elements in the PE array communicate with each other and the exact pattern of intercommunication is application-dependent.

A study of fault-induced errors in an associative processor shows that they are not easily detectable since a single fault may cause an arbitrary number of errors. This is evident for faults in global subsystems of Figure 1, such as the input and mask registers. A single fault in one processing element may cause errors in others because of PE intercommunication. The problem is further compounded by the fact that each PE performs logic and selective write operations on individual data bits which as we know are not easily checkable without a high level of redundancy.

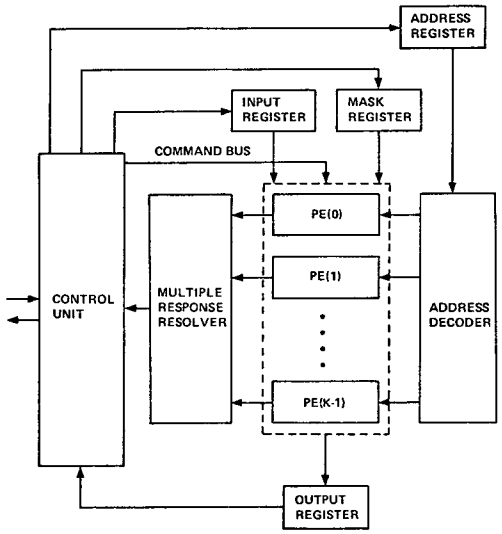

Figure 1. Associative Processor Model

The associative processor of Figure 1 can be made fault tolerant by dividing the PE array into identical modules which share spares. Let us assume that we have M modules, each consisting of P processing elements. It is possible to distribute the decoding and response resolution functions among the modules in order to reduce the complexity of the non-array portion to a minimum. Figure 2 shows the modules and their interconections. One-dimensional intercommunication between modules has been assumed for simplicity.

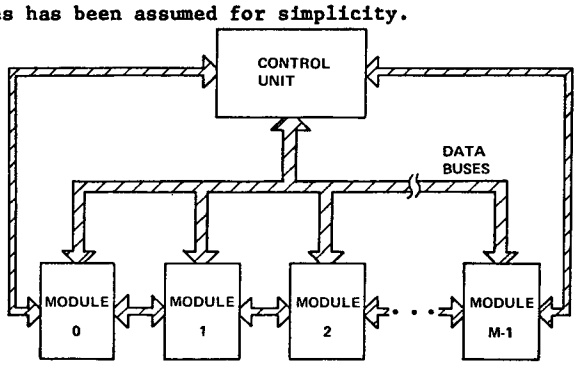

Figure 2. Modularized Associative Processor

Given a modular associative device as shown in Figure 2, it can be made fault tolerant by the following steps: (1) Incorporating internal failure detection ability within each module; (2) Adding S spare modules; and (3) Designing switching mechanisms and corresponding algorithms for reconfiguration. We will assume that the M + S operating and spare modules are permanently connected to the main data buses and that special isolating circuits exist between each module and the data buses. Therefore, reconfiguration takes place by "power switching" and by providing alternate intercommunication paths between modules.

DETECTION OF MODULE FAILURES

We first discuss the problem of error detection in associative processors with respect to the four classes mentioned previously. Then we will consider a technique which is applicable in all cases.

A fully parallel associative memory with only exact-match search operation and without masking capability can be protected against storage errors by using a code with a minimum distance of k in conjunction with mismatch detectors with a threshold of k. With this scheme, stored words containing k-1 or fewer errors will never respond to a search operation and are effectively isolated from the rest of the system until periodic diagnosis routines detect their failure. The difficulty is that such an associative device will have no application besides simple table look-up. For most other applications, masking capability, more complex search types, and arithmetic operations are essential.

Considerations for bit-serial systems are similar to those for fully parallel systems. One advantage which exists here is the serial processing of bits in each word. This allows us to artificially extend each operation to the entire word by performing "null" operation on bit positions not originally specified. Now, since all the bits of each word are processed serially, codes with low-cost serial encoding and decoding can be used to protect against storage errors. It should be noted, however, that if operations on small fields within the words are to be performed frequently, the above scheme may result in a significant reduction of speed.

As noted earlier, because processing is performed serially in a word-serial system, protection against failures becomes relatively simple. Low-redundancy coding can be used to protect against storage errors. Failures in the processing logic may be detected through self-checking [7] design. Self-checking translators may be needed to convert the storage encoding (S-encoding) to an encoding suitable for processing (P-encoding). The main requirement on the P and S encodings is that fast (parallel) translation between the two must be possible.

One favorable property of block-oriented systems with respect to fault tolerance is that during each operation cycle, a processing element operates on the entire block of information assigned to it. This enables the use of block codes which result in relatively low redundancy and have simple serial checking algorithms. If mechanical storage devices are used to implement such devices, error bursts become very probable due to dust particles, minute scratches, or defects in the oxide coating. It has been noted that low-cost arithmetic error codes are very effective for coping with such burst errors [8].

As can be seen from the previous discussion, low-redundancy coding techniques are applicable only in special cases. Design of logic circuits in self-checking form [7] (i.e., in a way that internal circuit failures manifest themselves on the circuit's output) particularly if 1-out-of-2 encoding is used, appears to be promising. However, because of the relatively higher complexity of the self-checking design approach as compared to low-redundancy coding techniques, this approach should be used when others fail or for protecting the system's hard core. A detailed discussion of self-checking design concepts is beyond the scope of this paper [5].

RECONFIGURATION THROUGH SWITCHING

For a modular associative device to tolerate module failures, the module interconnections should not be rigid as shown in Figure 2. Rather, the modules should be interconnected through specially designed switching cir-

cuits which prevent a system failure as a result of the failure of a module. The setting of these switching mechanisms determines the system configuration and can be changed by a central monitor if required. If a module error is indicated and the existence of a permanent failure is determined, reconfiguration procedures must be initiated to establish a new working configuration. In general, data transfers between modules and correction of fault-induced errors are needed as part of the reconfiguration process

We will assume only unidirectional (left to right) data flow between the modules in Figure 2. The generalization of the results to bidirectional data exchange is straightforward. After detecting the existence of a faulty module, the following steps must be taken before normal operation can resume: (1) Locating the faulty module; (2) Determining a new working configuration; (3) Initiating appropriate data transfers; and (4) Effecting reconfiguration through switching. The criteria that should be used in evaluating each reconfiguration scheme include: (1) The amount of data transfers needed; (2) The complexity of the reconfiguration algorithms; (3) The number of spares S needed for tolerating f module failures; and (4) The complexity of additional switching circuitry.

We first discuss centralized reconfiguration schemes in which the switching hardware is external to the modules. A straightforward solution is the use of a "permutation network" [9] which can interconnect the modules in any order. Such a permutation network can be implemented as a cellular array [10] of two-state basic modules. Since the complexity of such a cellular permutation network is roughly proportional to the square of the number of modules, its use can be justified only if a relatively small number of modules are involved. The two-state basic modules can be used in a different way to form a "shorting network" [9]. As shown in Figure 3, such a shorting network can be used to route data around the faulty and spare modules. One disadvantage of this scheme, particularly as shown in Figure 3, is the excessive amount of data transfers needed in the case of a failure. The number of transfers needed can be reduced by optimal placement of the spare modules [5].

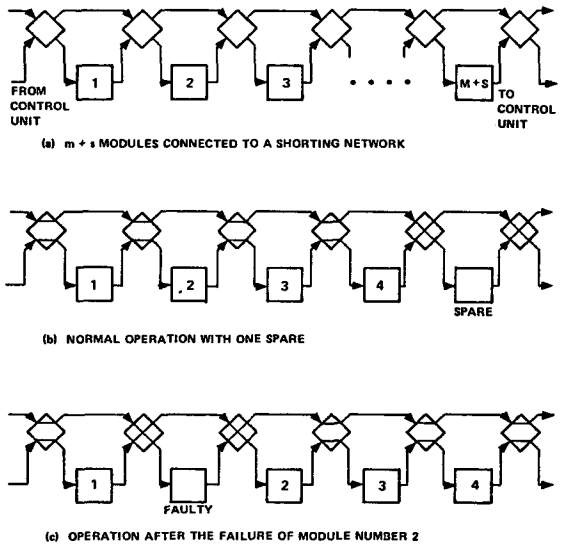

Figure 3. Reconfiguration with a Shorting Network

Another approach to the reconfiguration problem is the use of a distributed switching mechanism; i.e., distributing the switching hardware among the modules. This can be done by providing each module with a set of input and output lines instead of one as shown in Figure 2. Then if a successor module connected to one module output fails, a module connected to another output can act as its successor. The simplest case, which will be discussed here, is when each module has two sets of inputs and two sets of outputs. The two inputs and two outputs are distinguished by the letters H and V (horizontal and vertical). The module has four states denoted by HH, HV, VH, and VV, depending on whether the H or V input is used and whether the output is generated on the H or V output.

Figure 4 shows a two-dimensional arrangement of the basic modules. It can be seen in Figure 4 that the 9 modules can be connected into a string. If any single module fails, the remaining 8 can continue their operation. Double module failures will leave at least 6 usable modules. Hence, with M=8 and S=1, this scheme can tolerate all single module failures. With M=6 and S=3, all double failures can also be tolerated as well as some triple failures. The problem of optimal interconnection patterns for the tolerance of a maximum number of module failures has not been solved. The basic advantage of this scheme is that the switching mechanism is not part of the system's hard core since a failure in the switching circuits is equivalent to a module failure. The main disadvantages of this scheme are the complexity of the reconfiguration algorithm, excessive data transfers, and tolerance of fewer than S failures.

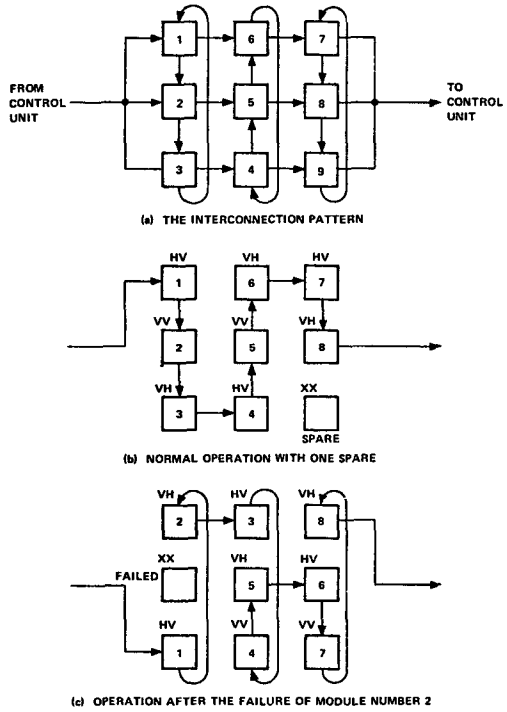

Figure 4. An Example of Distributed Reconfiguration

A CASE STUDY

In this section, we illustrate the applicability of some of the techniques discussed previously by presenting the design and evaluation of a fault-tolerant associative processor called SPARE (inverse acronym for Error-tolerant and Reconfigurable Associative Processor with Self-repair). SPARE is essentially a fault-tolerant version of an associative processor which has been described previously [3]. Figure 5 shows a block diagram of the non-redundant system. The random-access memory is used for storing instructions and constants and consists of 4096 24-bit words. The associative memory contains 512 96-bit words.

The non-redundant associative processor of Figure 5 can be divided into two parts: (1) The associative (parallel) section, which consists of the associative memory array, bit column selection logic, and word logic; (2) The control and sequencing (sequential) section,

which contains all other subsystems of Figure 5. The sequential section uses status signals and test inputs for monitoring the operation of the parallel section. We now briefly discuss the three main features of SPARE; i.e., error tolerance, reconfigurability, and self-repair.

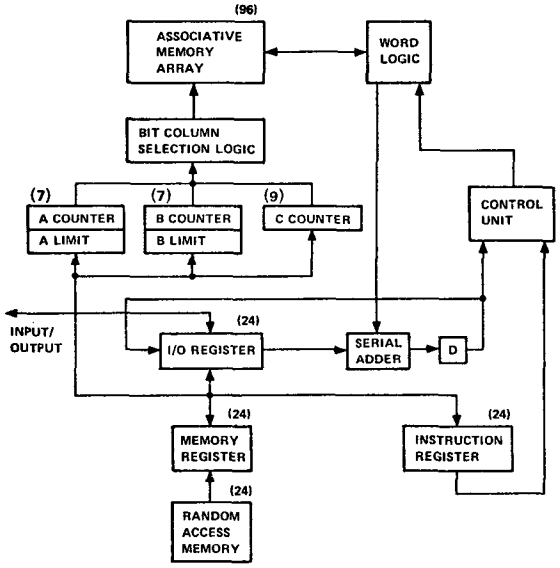

Figure 5. The Non-Redundant Associative Processor

To achieve error tolerance, the parallel section of SPARE is divided into M identical modules. S spare modules are shared by the operating modules. Each module has internal failure detection capability which is provided by self-checking design of its circuitry using two-rail encoding of logic variables. When a module error is indicated to the sequential section, the recovery mode is entered and the final result may be the replacement of the faulty module by a spare module. The sequential section of SPARE resembles a small general-purpose computer and can, therefore, be made fault tolerant by conventional techniques.

One of the very important properties of associative processors is simple modular growth. The size of an associative processor can grow without a need to alter its algorithms. This suggests that if additional processing capability is required, the redundant processing logic in SPARE can be utilized. Even the two channels of the two-rail circuits can be used independently to double the processing capability if certain design criteria are met [5]. Specifically, we postulate the following operation strategy for SPARE: (1) During normal operation the system works in redundant mode with a number of spare modules; (2) If a module failure occurs or additional processing capability is needed and if a sufficient number of spares are available, they are switched in; (3) If a module failure occurs or additional processing capability is needed and spare modules are not available, the system reconfigures into simplex mode by utilizing the two channels of the two-rail circuits independently.

Of the reconfiguration techniques discussed in the previous section, the one using a permutation network seems to be suitable for SPARE since only one intercommunication line (two in self-checking design) exists between modules and the number of modules is expected to be small (M=4 or 8, for example). The self-repair process will then essentially consist of computing and setting of a new state for the permutation network. This process must be followed by a recovery procedure to transfer the data stored in the failed module to the one which replaces it. The permutation network has a two-rail self-checking design but no spare is provided for it.

In computing the reliability of SPARE, we will assume that the coverage factor C includes the reliability of the permutation network. Using the reliability modeling technique of Bouricius et at [11], we find the reliability improvement factor defined as $[1-R_{nr}(T)] \div [1-R_r(T)]$ as a function of mission time T for several configurations of SPARE (R_{nr} and R_r denote the non-redundant and redundant reliabilities, respectively). Figure 6, which depicts the resulting curves, shows that for mission times which are short compared to the MTBF for the non-redundant system, a significant increase in reliability is possible with a low levle of modularization and a relatively small number of spare modules.

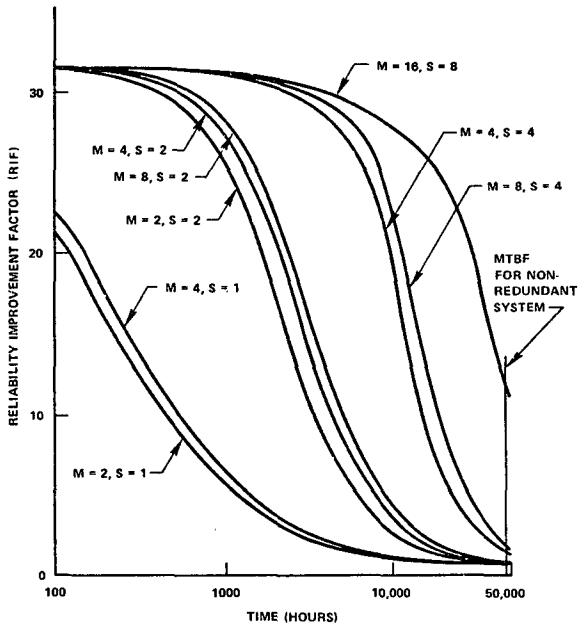

Figure 6. Reliability Improvement Factor for SPARE (C=0.99).

CONCLUSION

In this paper, we have presented some results of a study on the fault tolerance of associative processors. Our main conclusions are as follows:

(1) Dynamic redundancy is to be preferred over static approach because associative processors lend themselves naturally to modularization and since spares can be shared by a number of identical modules.

(2) Low-redundancy coding techniques are applicable for error detection in associative processors but only in special cases. In particular, the use of arithmetic error codes for block-oriented systems appears to be promising.

(3) Application of self-checking circuit design techniques seems to be an attractive alternative for error detection in associative devices.

(4) Complex switching mechanisms and algorithms need to be devised to enable the sharing of spares by a collection of identical modules which communicate with each other.

Further research is needed in two equally important areas. The first area is the design of completely checked digital circuits. Systematic techniques need to be developed to aid the designers in choosing suitable input and output encodings and producing a self-checking design when presented with a non-redundant circuit or its functional behavior. The second area deals with general techniques for reconfiguration in array processors. Extension and generalization of the results presented here are possible in two directions. First, one can conceive of other interconnection schemes for

the case where one-dimensional intercommunication exists between modules. For example, we may consider a three-dimensional interconnection pattern in which there are three choices for each of the left and right neighbors for a module. Second, one may seek generalizations to the cases where multi-dimensional module intercommunication is used. This is a considerably more complex problem.

REFERENCES

[1] Thurber, K.J., "An Associative Processor for Air Traffic Control," AFIPS Conference Proceedings, Vol. 38 (1971 Spring Joint Computer Conference), AFIPS Press, Montvale, New Jersey, 1971, pp. 49-59.

[2] Berg, R.O. and M.D. Johnson, "An Associative Memory for Executive Control Functions in an Advanced Avionics Computer System," Proceedings of IEEE International Computer Group Conference, June 1970, pp. 336-342.

[3] Ewing, R.G. and P.M. Davies, "An Associative Processor," AFIPS Conference Proceedings, Vol. 26 (1964 Fall Joint Computer Conference), Spartan Books, Baltimore, Maryland, 1964, pp. 147-158.

[4] Proudman, A., "Bulk Associative Memory with Error Correction," IBM Technical Disclosure Bulletin, Vol. 12, No. 7, pp. 1076-1077, December 1969.

[5] Parhami, B., "Design Techniques for Associative Memories and Processors," Technical Report UCLA-ENG-7321, Computer Science Department, University of California, Los Angeles, March 1972. (Also published as a Ph.D. dissertation).

[6] Parhami, B., "Associative Memories and Processors: An Overview and Selected Bibliography," Proceedings of the IEEE, Vol. 61, No. 6, pp. 722-730, June 1973.

[7] Carter, W.C. and P.R. Schneider, "Design of Dynamically Checked Computers," Information Processing 68, (Proceedings of IFIP Congress, Edinburgh, Scotland, August 1968), North Holland Publishing Company, Amsterdam, 1969, pp. 878-883.

[8] Parhami, B. and A. Avižienis, "Application of Arithmetic Error Codes for Checking of Mass Memories," Digest of International Symposium on Fault-Tolerant Computing, Palo Alto, California, June 1973, pp. 47-51.

[9] Levitt, K.N., M.W. Green, and J. Goldberg, "A Study of Data Commutation Problems in a Self-Repairable Multiprocessor," AFIPS Conference Proceedings, Vol. 32 (1968 Spring Joing Computer Conference), Thompson Book Company, Washington, D.C., 1968, pp. 515-527.

[10] Kautz, W.H., K.N. Levitt, and A. Waksman, "Cellular Interconnection Arrays," IEEE Transactions on Computers, Vol. C-17, No. 5, pp. 443-451, May 1968.

[11] Bouricius, W.G., W.C. Carter, and P.R. Schneider, "Reliability Modeling Techniques for Self-Repairing Computer Systems," Proceedings of the 24th National Conference of ACM, San Francisco, California, August 1969, pp. 295-309.