FAULT TOLERANCE TECHNIQUES FOR AVIONICS WAVEFRONT ARRAY PROCESSORS

Ralph Duncan
Control Data / Government Systems
300 Embassy Row
Atlanta, GA 30328
(404) 399-2105

Abstract

The paper analyzes fault tolerance techniques for wavefront array processing in an avionics environment, emphasizing techniques that exploit wavefront arrays' asynchronous character. Salient features are proposed, such as Array Manager software, dedicated fault report and test links, and programmable array module interconnections. Detection mechanisms for processor and interconnection network faults are discussed. A centralized, asynchronous approach to fault isolation is described. A three-level strategy for fault recovery is examined. First, programmable module interconnections are used for dynamically swapping processor modules from a pool of spares. Second, affected applications migrate to subsystems with spare processing capacity. Finally, applications are replaced with alternative software versions that utilize a faultless, contiguous subset of array module processing nodes.

Wavefront Arrays in an Avionics Environment

The most likely role for wavefront array processors in an avionics environment will be as part of a distributed system of heterogeneous architectures, in which wavefront arrays will perform matrix-oriented operations for signal and image processing applications.

Such an environment, as envisioned by a recent study,[1] would include subsystems composed of either standard, general-purpose processors (such as 1750As) or of specialized (parallel) processors driven by a standard processor 'host' (fig. 1). A wavefront array processor would be used as a powerful coprocessor for an avionics subsystem that is controlled by a largely autonomous operating system (OS) kernel, which executes on a standard processor and handles communications with other subsystems and with the OS executive. The local OS kernel would provide a software interface for controlling the wavefront array's parallel processing and would encapsulate all subsystem fault management operations that could be handled locally. Fault management that cannot be handled at the subsystem level entails passing messages to the OS executive, which coordinates activities that cross subsystem boundaries.

Wavefront Array Processor Definition

Wavefront array processors, first proposed by S.Y. Kung,[2] are characterized by the following fundamental characteristics:

- the computational model combines systolic data pipelining with asynchronous data-flow execution;
- modular processors are united by a regular, local interconnection network (IN);
- asynchronous handshaking, rather than a global clock and explicit time delays, synchronizes data pipelining from processor to processor.

Wavefront arrays coordinate pipelined data-flow in the following way. When a processor completes its current computations and is ready to pass data to a successor (a contiguous processor in the local interconnection network), it informs the successor, sends data when the successor indicates readiness, and receives an acknowledgment from the successor. This handshaking mechanism makes computational "wavefronts" pass smoothly through the array without intersecting, as the array's processors act as a wave propagating medium.

Figure 2 (following page) depicts a wavefront array that performs a 2x2 matrix multiplication by pipelining operands from node to node, as matrix cell results accumulate in appropriate processors. Thickened lines indicate the propagation of computational 'wavefronts.'

Example Wavefront Array Subsystem

The hypothetical wavefront array subsystem used below to illustrate relevant fault-tolerance techniques follows two Air Force PAVE PACE architecture studies[3,4]. The subsystem (figure 3) consists of a general-purpose processor 'host', multiple array processor modules, and off-module memory resources. Each module implements wavefront data pipelining with processors that are organized into a symmetrical mesh by a simple, local interconnection network. Each processor node has local memory sufficient to hold several short programs geared to wavefront fine-to-medium grained parallelism.

CH2929-8/90/0000-0344 $1.00

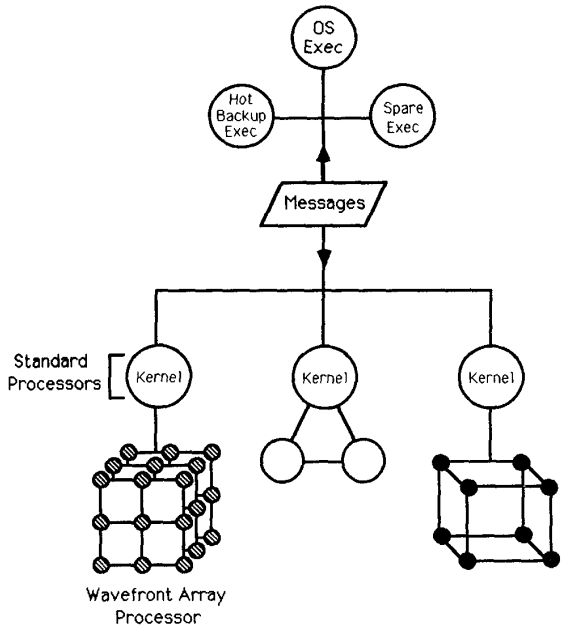

Figure 1. Distributed Avionics Environment

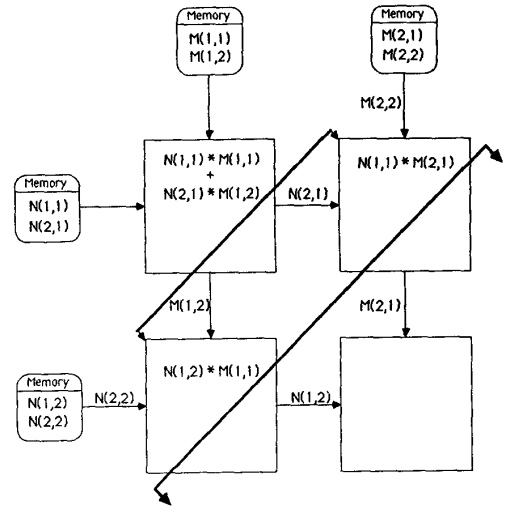

Figure 2. Wavefront Data-pipelining

An interconnectivity scheme that supports fault tolerance is described below. Programmable crossbar switches can be used to connect the processors along wavefront array module edges to processors situated at the same relative position on the opposite edges of other cards (figure 4). Following wavefront and systolic array restrictions, only these processors along a module edge can communicate with off-module memory.

In addition, a pair of high-speed, serial control lines connect each array processor node to the subsystem's host processor; this allows the host-based kernel software to selectively engage individual array nodes in fault management operations while unaffected nodes continue normal operations.

Envisioned Subsystem Component Interaction

The wavefront array modules will likely function as a powerful matrix calculation coprocessor for applications executing on the host processor. Host-resident Array Manager system software can control the wavefront array modules for an application and provide structured fault management services. Applications could order the Array Manager to program wavefront module interconnections and initialize specified, pre-stored programs in the array nodes. Subsequently, array modules would read data from off-module memory, pipeline the data through the array and write accumulated results back to off-module memory.

Wavefront Fault Detection Mechanisms

Execution-time faults can be detected by the following means:

- parity-checking,
- data time-out functions,
- fault reports,
- pre-stored module tests.

Transient IN communication faults can be detected by parity checking firmware in a receiving node, which responds with a resend-data request. If such faults exceed a specified threshold, the receiving node can recognize the possibility of a hard fault and can notify the Array Manager that a particular node and IN link are possibly faulty. In the hypothetical system, this notification is achieved by using the dedicated control lines to the host processor for transmitting a terse 'fault report.' Such fault signaling causes an interrupt on the host that transfers control to an appropriate Array Manager (AM) routine. The AM then takes charge of the fault isolation process.

Similarly, faults can be detected by wavefront array nodes and reported to the Array Manager when a neighboring node does not complete data-transfer handshaking protocols or upon a time-out due to a neighbor not transmitting data within an acceptable time frame.

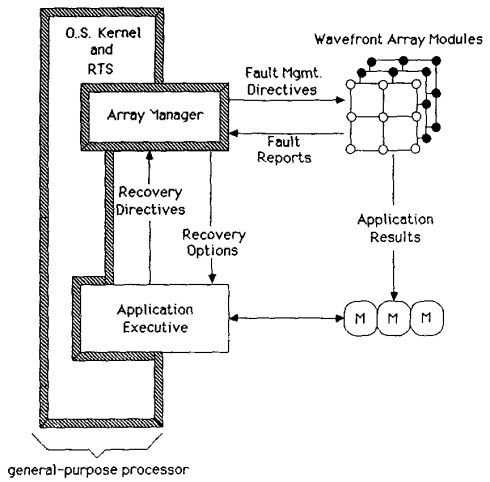

Figure 3. Example Wavefront Array Subsystem

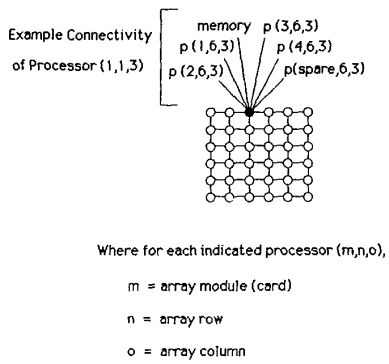

Figure 4. Module Edge Interconnectivity

Asynchronous data pipelining between adjacent nodes, which is a central feature of wavefront arrays, allows neighboring processor nodes to detect communication-related faults. However, some processor faults cannot be detected by such means. These faults can be uncovered by module-level tests and by individual array node self-test. A pre-stored test that exercises each module processing node can be run at system start-up time or periodically to detect such processor faults. The use of self-tests for individual nodes is discussed below in conjunction with fault isolation.

Wavefront Fault Isolation Techniques

A variety of fault isolation techniques can be exploited by wavefront arrays, including:

- directed self-test of individual nodes,
- fault report pattern analysis,
- encoded, module-level tests.

The Array Manager attempts to isolate faults when it receives one or more fault reports indicating that reporting nodes do not receive a handshake from a neighboring node, repeatedly receive data from a neighbor that fails parity checks, or do not receive data/handshakes from a neighbor before the time-out threshold is reached. The Array Manager then uses control lines to force the suspect node off-line and request a self-test. If the node then fails the self-test or fails to report self-test results, the node is regarded as faulty. Self-test directives and results could flow through the array's processor-to-processor IN, at the cost of complicating inter-processor communications.

Analyzing the spatial pattern of nodes generating fault reports can aid fault isolation. An IN link fault can be distinguished from a processor fault, for example, when two nodes that are unable to communicate properly both pass self-test and are not the object of fault reports generated by other neighboring nodes.

Similarly, a faulty node that incorrectly indicates its self-test is successful can be isolated by recognizing that multiple neighbors continue to generate fault reports for it.

Note that utilizing fault report pattern analysis may require suppressing time-out fault reporting when a suspect node is forced off-line in order to suppress misleading reports directed at nodes that are waiting for data from the node undergoing self-test.

Finally, processor faults can be isolated by using a pre-stored, module-level test program, which has been encoded in a manner that allows a faulty node to be isolated by comparing stored correct results with actual results[5].

Fault-Recovery Techniques

Three levels of fault recovery options are

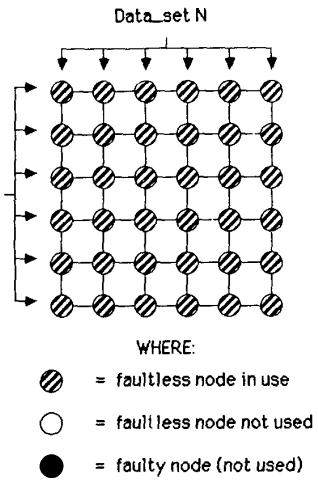

Figure 5. Data-flow in a Faultless Module

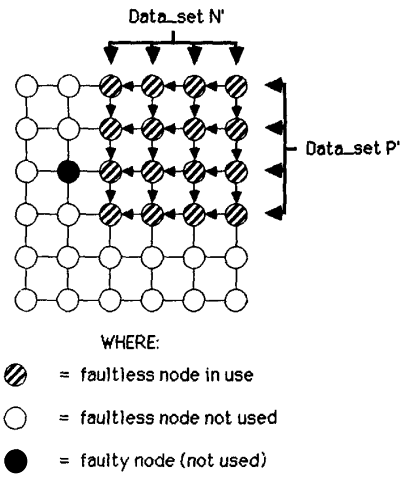

Figure 6. Altering Data-flow Direction

available for multiple-module wavefront arrays:

- dynamic reconfiguration,
- application migration,
- alternative application version mapping.

In order to allow application-specific fault recovery, the Array Manager can respond to hard faults by interrupting host-resident application code, describing the fault type and available recovery options, and acting on application requests to attempt particular recovery options.

Dynamic Reconfiguration

Dynamic hardware reconfiguration can be achieved by either swapping in a faultless module from a pool of spares or by reconfiguring the affected module's interconnection network.

The programmable crossbar connections sketched for the example subsystem would be programmed at application initiation to yield a logical inter-module topology appropriate for the application (e.g., a linear pipeline, a torus). This scheme would allow spare swapping by reprogramming the module edge processor connections that currently involve the faulty module. Current limitations of crossbar technology might severely constrain the number of spares that could be accommodated in this fashion. Alternatively, bus interconnection between modules would potentially allow a large number of spares, given acceptable inter-module communication speeds.

A wide variety of intra-module array reconfiguration schemes have been proposed[6]. For example, set-switching schemes typically use spare processor sets (such as rows or columns) and switches to replace a faulty set with a spare. Alternatively, processor-switching approaches, such as fault-stealing techniques, may utilize sophisticated algorithms and INs to effectively replace individual faulty nodes.

Application Migration

If no spare modules are available for reconfiguration, an attempt can be made to have the affected application 'migrate' to a subsystem that can execute the application while meeting its performance and service deadline requirements.

In a distributed avionics environment, application migration would likely entail the local kernel sending a Migration Request message to the OS executive, which would determine whether other subsystems could accommodate the application's resource and time deadline requirements with both their current applications and the new application running. The Executive would choose where to move the affected application on the basis of system-wide load-balancing criteria.

Alternative Application Version Mapping

When spare wavefront array modules are not available and the application cannot migrate to

another subsystem with adequate processing power to spare, alternative versions of the affected application could be executed that:

- use a faultless subset of the faulty module,
- trade reduced throughput for continuing to execute with reduced processing resources.

When a simple processor-to-processor IN that cannot switch around faulty nodes is used, the alternative version recovery approach is tantamount to finding a faultless, contiguous subset of the faulty module and mapping the relevant alternative application version to it.

Techniques[7] have been developed to map algorithms to data-pipelined arrays that have fewer processors than the algorithm requires for results accumulation. Such techniques typically partition the algorithm into subsets that match the number (and topology) of processors available in the physical array. These techniques could be used to automate the production of alternative application versions sized to appropriate subsets of the wavefront array.

The practical exploitation of alternative application versions for fault tolerant reconfiguration of a wavefront array will involve:

- choosing appropriate versions to pre-store,
- dynamically selecting and mapping a version,
- handling changes in pipelined data flow.

Producing and storing an alternative version for every possible size of faultless subset may not be tractable for a large array module (especially if the subset need not be symmetrical). Thus, a restricted subset of possible versions must be chosen. Similarly, storing a single version for a given mesh subset size and mapping it to multiple mesh locations is preferable to storing programs for each possible faultless subset permutation. Hence, system software must perform topological analysis of the faulty node or link's location in order to determine the optimal alternative version (size) to use and the location to which it must be mapped.

Finally, provision must be made for changing data flow to and from off-module memory. Existing algorithms often read such data with processors situated along two adjacent module edges (as shown in figure 2). Using an optimally-sized alternative version, however, may require altering data-flow direction. Figure 5 shows a module executing an algorithm that accumulates results for a 6x6 matrix multiplication. Off-module data enters along the top and left module edges. The data entering along a given edge flows toward the opposite edge as it passed from node to node.

In figure 6, the direction of data-flow has been changed to accommodate the alternative application version that utilizes the largest contiguous subset of the module that is both faultless and symmetrical. Changing data-flow direction in this manner requires parameterizing the directions that node programs use for data pipelining.

Note that alternative application versions could use asymmetrical, faultless mesh subsets. It may also prove tractable to concurrently run multiple versions to cooperatively perform the original application task; that approach might successfully adapt memory management mapping algorithms.

Conclusions

A fundamental characteristic of wavefront arrays, controlling data pipelining through asynchronous handshaking, allows them to exploit a wide variety of fault tolerant techniques. Asynchronous control supports conducting effective fault detection and isolation operations with minimal performance degradation, since individual nodes can issue fault reports or undergo off-line testing without disrupting the entire system. Dynamic module swapping should prove feasible, since the physical proximity of modules is not important when a global clock is not used for synchronization. If a fault-tolerant IN is used, well-known reconfiguration techniques can be used. An array using a simple, point-to-point IN could pre-store alternative application versions that can be readily mapped to faultless, contiguous subsets of a module to provide a high degree of robustness. Wavefront arrays are attractive architectures for avionics systems, since they offer a high degree of scalable parallelism and can often localize performance penalties of fault-tolerant operations.

References

1. Ralph Duncan and Robert Hanna, "A Fault-Tolerant O.S. Kernel for Avionics Wavefront Array Processing," Proc. 42nd Natl. Aerospace and Electronics Conf. (Dayton, May 21-25), 1990, pp. 227-232.
2. S.Y. Kung, S.C. Lo, S.N. Jean, and J.N. Hwang, "Wavefront Array Processors - Concept to Implementation," Computer, Vol. 20, No. 7 (July 1987), pp. 18-33.
3. Control Data Corporation, "Fault Tolerant System Control Study," Wright Research and Development Center Tech. Report, WRDC-TR-90-1002, March 1990.
4. Westinghouse Electric Corporation, "Parallel Processor Application Study," Tech. Report, WRDC contract F33615-88-C-1712, 1989.

5. J.A. Abraham, P. Banerjee, C-Y. Chen, W.K. Fuchs, S-Y. Kuo and A.L. N. Reddy, "Fault Tolerance Techniques for Systolic Arrays," Computer, 20/7, July 1987, pp. 65-74.

6. M. Chean and J.A.B. Fortes, "A Taxonomy of Reconfiguration Techniques for Fault-Tolerant Processor Arrays," Computer, 23/1, Jan. 1990, pp. 55-69.

7. J. Navarro, J. Llaberia and M. Valero, "Partitioning: An Essential Step in Mapping Algorithms Into Systolic Array Processors," Computer, Vol. 20, No. 1, July 1987, pp. 77-89.