Fault-Tolerant Parallel Processor

Richard E. Harper and Jaynarayan H. Lala
Charles Stark Draper Laboratory, Cambridge, Massachusetts 02139

Future mission-critical computing systems are likely to have throughput requirements that current fault-tolerant computers cannot meet, whereas currently available high-throughput computers may have inadequate reliability for such applications. This paper addresses issues central to the design and operation of an ultrareliable, "Byzantine resilient" parallel computer. Interprocessor connectivity requirements are met by treating connectivity as a resource that is shared among many processing elements, allowing flexibility in their configuration and reducing complexity. Redundant groups are synchronized solely by message transmissions and receptions, which also provide input data consistency and output voting. Reliability analysis results are presented that demonstrate the reduced failure probability of such a system. Performance analysis results are presented that quantify the temporal overhead involved in executing such fault-tolerance-specific operations. Empirical performance measurements of prototypes of the architecture are presented.

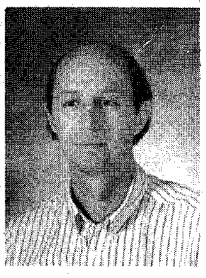
I. Introduction

FUTURE mission- and life-critical computing systems are likely to have throughput requirements that current fault-tolerant computers cannot meet, whereas currently available high-throughput computers may have inadequate reliability for such applications. Typically, the allowable failure probabilities range from 10^{-4} to 10^{-6} per hour for mission-critical functions and 10^{-6} to 10^{-10} per hour for vehicle-critical and crew-safety functions. A second major need is a high level of computer performance. This not only includes high throughput and large memories, but also requirements imposed by the operational environment. For example, the computer architecture should be capable of adapting itself to the varying requirements of a mission in real time by trading performance for reliability and vice versa.

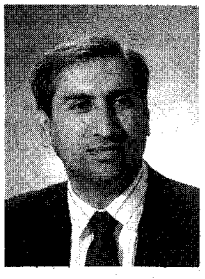
Three representative applications requiring this combination of high throughput and reliability are currently under active development at the Charles Stark Draper Laboratory (CSDL). The first is an adaptive tactical navigation system, which is one of the functions performed by the pilot's associate for fighter aircraft. Components of this mission-critical application consist of knowledge-based expert systems that we have demonstrated can be efficiently sped up via parallel processing. A second application comprises control, diagnostics, sensor-redundancy management, performance seeking, and failure accommodation for an advanced fighter aircraft engine. It is anticipated that several processors will be required to perform this application. The third application consists of integrated flight and propulsion control for a short takeoff and vertical landing (STOVL) fighter aircraft. Again, the preliminary throughput estimates indicate the need for several processors to perform this flight-critical function.

Two key figures of merit for comparing different architectures are performance, as indicated by the throughput, and reliability, as indicated by the probability of failure per hour. Figure 1 shows the performability, i.e., the performance vs reliability, of existing fault-tolerant computers and high-throughput computers. Superimposed on this diagram is an estimate of the performability likely to be required for future applications. Existing fault-tolerant computers fall short of the anticipated throughput needs of such applications by one order of magnitude, while existing high-speed computers appear to fall short of the reliability goals by several orders of magnitude.

This paper discusses a computer architecture, called the fault-tolererant parallel processor (FTPP), that has been



Richard E. Harper received his B.A. in Physics and M.S. in Aeronautics and Astronautics from Mississippi State University in 1976 and 1977, respectively. He received his Ph.D. from the Massachusetts Institute of Technology in 1987. Since 1983, he has worked in the Fault-Tolerant Systems Division of the Charles Stark Draper Laboratory, Cambridge, MA. His technical interests lie in the areas of reliable computing and communication systems. He is a Member of AIAA.



Jaynarayan H. Lala has been the Division Leader of the Fault-Tolerant Systems Division at the Charles Stark Draper Laboratory in Cambridge, MA, since July 1985. His research interests include design, evaluation, and validation of fault-tolerant architectures for high-integrity systems. He received the Ph.D. in Instrumentation and the M.S. in Aeronautics and Astronautics from the Massachusetts Institute of Technology in 1976 and 1973, respectively. He received the B.S. degree in Aeronautical Engineering from the Indian Institute of Technology, Bombay, India, in 1971. He is an Associate Fellow of the AIAA.

Received Nov. 30, 1989; revision received May 1, 1990; accepted for publication May 2, 1990.

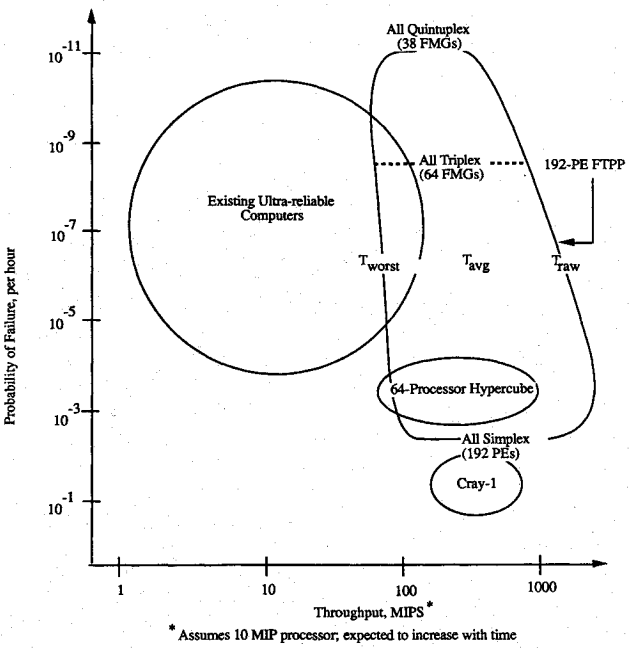


Fig. 1 Performability diagram.

Table 1 Desired attributes in fault-tolerant parallel processing

Desired attribute	Approach
High throughput	MIMD parallel processing
High reliability	Fault tolerance through parallel-hybrid redundancy
Reconfigurability	Functional synchronization, shared connectivity
Mixed redundancy	Correct Byzantine-resilient inter-FMG communications protocols
Heterogeneity	Standard interface to FT-specific hardware; functional synchronization
Efficiency	Shared connectivity; hardware implementation of fault-tolerance-specific functions
Upgradeability	Standard interface to FT-specific hardware; functional synchronization
Ease of implementation	Build only fault-tolerance-specific hardware; off-the-shelf hardware and software
Programmability	Off-the-shelf hardware, software; Byzantine resilient virtual circuit
Analyzability	Framework of Byzantine resilience theory
Validatability/verifiability	Correct Byzantine-resilient "hard core"

designed by combining the disciplines of fault-tolerant (FT) computing and parallel processing to yield a theoretically sound high-throughput and high-liability computer.

II. Objectives and Approach

The assertion is often made that parallel processors are intrinsically reliable, fault tolerant, and reconfigurable due to their multiplicity of processing resources.[1] In fact, the only intrinsic attribute guaranteed by multiple processors is a higher total failure rate. The synergism between parallel processing and fault tolerance is very likely to be destructive and result in an unreliable and low-performance system, unless exploited with due regard to sound theoretical principles of fault tolerance. Therefore, it is useful to enumerate at the outset the desirable characteristics of a parallel processing approach to high-reliability high-throughput computing. Table 1 shows the desired attributes and our approach to providing these attributes in the FTPP. The details of the approach are described in the remainder of the paper. The salient features of the approach are as follows.

The classical multiple-instruction multiple-data (MIMD) approach implemented using a multiplicity of processing elements (PE's) provides high throughput (see Ref. 2 for a summary of parallel processing). In the FTPP, these PE's are operated as redundant groups to provide high reliability. Inter-PE connectivity, which is an expensive resource and contributes to increased failure rate and complexity, is shared among many PE's. This, in conjunction with hardware implementation of fault-tolerance-specific functions, keeps the overheads of fault tolerance small. The shared connectivity, in conjunction with functional synchronization of processes executing on redundant PE's, makes the system highly reconfigurable. PE's can be operated in parallel redundant groups with each redundant group executing a different process. In case of a PE failure in one group, a PE from another group, within certain constraints, can be used to replace the failed PE. This form of redundancy, called parallel-hybrid redundancy, allows the FTPP to tolerate a number of sequential failures and increases the mission time during which a high reliability can be maintained without repairs.

By conforming to certain simple requirements, the FTPP can tolerate any arbitrary hardware failure mode. This feature, known as Byzantine resilience, not only guarantees high reliability, but it also provides a number of other attractive attributes. For example, the framework of the Byzantine resilience theory makes the system amenable to analytical modeling techniques and to systematic techniques for validating the system implementation.

Systems functions typically vary in their degree of criticality. For example, flight control is more critical than mission planning, in that the loss of the former means vehicle loss while the loss of the latter implies mission abort. It is therefore desirable to have an architecture that supports such mixed reliability requirements, while at the same time preventing functions executing at a lower reliability level from corrupting those functioning at a higher reliability level. The mixed redundancy attribute of the FTPP is intended to allow the user to match function criticality with the appropriate redundancy level, thus making the most efficient use of processing resources. Although different processor groups may have different redundancy levels, they communicate with each other reliably in the presence of failed processors by using protocols that prevent a lower redundancy level group from inhibiting the correct functioning of a higher level group.

Finally, the most challenging and notorious aspect of parallel processors as well as fault tolerant computers has been the great difficulty encountered by users in writing application programs for them. A computer that combines parallel processing and fault tolerance has all of the necessary ingredients to turn it into a user-hostile machine. The redundancy management and fault-tolerance features in the FTPP have been implemented such that they are totally transparent to the user. By providing certain guarantees, collectively known as the Byzantine resilient virtual circuit (BRVC) abstraction, a very simple virtual architecture is presented to the applications programmer that hides the complexity associated with redundant groups of processors communicating internally within the group and externally with other processor groups. Table 1 summarizes additional desirable attributes in a fault-tolerant parallel processor, along with the FTPP approach to providing each attribute.

III. Theoretical Approach to Achieving Fault Tolerance

It is our contention that for a computer to be considered adequately reliable for life- or mission-critical applications, it must be capable of surviving a specified number of component failures with a probability approaching unity.[3] A conservative failure model is to consider failures as consisting of arbitrary behavior on the part of failed components. This type of fault, known as a Byzantine fault, may include stopping and then

restarting execution at a future time, sending conflicting information to different destinations, and other malfeasance.

Since the concept of Byzantine resilience is central to the theory and operation of the FTPP, it is important to discuss the motivation for this seemingly extreme degree of fault tolerance. Cost-effective validatability and achievement of high reliability comprise an important subset of motivating factors.

Validation-based motivation for Byzantine resilience is perhaps best viewed in the context of an example. We suppose that a digital computer system having a maximum allowable probability of failure of 10^{-9} per hour is required, and that this system must be constructed of replicated channels each of which has an aggregate failure probability of 10^{-4} per hour[4]. Consider a traditional failure modes and effects analysis (FMEA)-based approach to achieving the requisite failure rate: likely failure modes of the system are analyzed, predictions of their likely extent and effects are made, and suitable fault tolerance techniques are developed for each failure mode that is considered to possess a reasonable chance of occurring. For this approach, it is clear that for the system to meet the reliability requirement, the probability that any given fault is not covered must be less than $\approx 10^{-9}/10^{-4} = 10^{-5}$; that is, it is necessary that the likelihood of a failure occurring which was not predicted, planned for, and a suitable fault-tolerance technique devised must be less than $\approx 10^{-5}$. Viewed another way, it is (or should be) incumbent upon the designer to prove to an aggressive and competent inquisitor such as a certification authority, that fewer than one in 100,000 faults that could occur in the field (as opposed to those induced or injected in the laboratory) could conceivably defeat the proposed fault-tolerance techniques. If this cannot be demonstrated within a reasonable amount of time and money, then it is in turn infeasible to validate the FMEA assumptions and hence the claimed 10^{-9} per hour failure rate.

The FMEA process is tedious, time-consuming, and extremely expensive. This is attested to by the seemingly contradictory trend of increasing costs of digital avionics systems even as the cost of hardware continues to decline. This is at least partially because the cost of validating critical systems is completely overwhelming the cost of their design and construction. Software validation is a major component of this cost, and inappropriate fault-tolerance-related architectural features only aggravate its difficulty.

In contrast, consider another fault tolerance technique that guarantees that the system can tolerate faults, without relying upon any a priori assumptions about component misbehavior. In effect, a faulty component may misbehave in any manner whatsoever, even to the extreme of displaying seemingly intelligently malicious behavior. A system tolerant of such faults would obviate the expensive and physically intractable problem of convincing a knowledgeable inquisitor of the validity of restrictive hypotheses regarding faulty behavior, in effect permitting faulty behavior to subsume all conceivable FMEA's. Such a system is denoted "Byzantine resilient," that is, capable of tolerating Byzantine faults. The source of this terminology may be found in the saminal literature on the theory of ultra-reliable distributed systems[5]:

> Reliable computer systems must handle malfunctioning components that give conflicting information to different parts of the system. This situation can be expressed abstractly in terms of a group of generals of the Byzantine army camped with their troops around an enemy city. Communicating only by messenger, the generals must agree upon a common battle plan. However, one or more of them may be traitors who will try to confuse the others. The problem is to find an algorithm to ensure that the loyal generals will reach agreement.

The generals in this analogy correspond to processors in a redundant computing system, the traitors correspond to faulty processors, and the messengers correspond to interprocessor communications links. It is typically assumed that faulty link (traitorous messenger) behavior is subsumed by faulty source processor (traitorous general) behavior.

One would expect a system capable of tolerating such a powerful failure mode to be intrinsically complex and possess numerous inscrutable and exotic characteristics. To the contrary, the requirements levied upon an architecture tolerant of Byzantine faults are relatively straightforward and unambiguous, simply comprising a lower bound on the number of fault-containment regions and their connectivity, their synchrony, and the utilization of certain simple information exchange protocols. We assert that satisfactorily demonstrating that an architecture possesses these simple attributes is far less expensive and time consuming than proving that certain uncovered failure modes can occur with a probability of at most 10^{-5}. Existing critical computing systems are typically designed to be triply or quadruply redundant anyhow; meeting the requirements for Byzantine resilience requires a simple rearrangement of the channels and addition of a few interchannel communication protocols. We think this minor rearrangement of the architecture recovers many times over the cost of an FMEA-based validation. Moreover, it is our experience that the run time overhead required to achieve Byzantine resilience can be substantially less than that required to achieve significantly lower levels of fault coverage using fault-tolerant techniques based on restrictive hypothetical models of failure behavior.

By making the system Byzantine resilient, we have imparted to it some powerful programming attributes that result in a significant reduction in software validation effort and cost. First, the hardware redundancy is transparent to the programmer. The applications programs and the operating system are developed, debugged, and validated in a simplex (nonredundant) environment without any regard for the redundant copies of the software executing on redundant hardware. Second, the management of hardware redundancy is transparent to the programmer. The applications programs as well as the operating system are rigorously separated from the hardware and software that manages redundancy. Redundancy management includes functions such as detection and isolation of faults, masking of errors resulting from faults, and reconfiguration and reallocation of resources. This rigorous separation allows independent validation of various software entities such as the applications programs, the operating system, and the redundancy-management software. By breaking the destructive synergism that comes from intertwining these entities, significant reduction in software validation effort has resulted for the FTPP's predecessors such as the fault-tolerant multiprocessor (FTMP),[6] the fault-tolerant processor (FTP),[7,8] and the advanced information processing system (AIPS).[9,10] Third, consistent interprocessor message-ordering and validating are guaranteed, even in the presence of arbitrary faults. This guarantee relieves the programmer from consideration of faulty behavior when designing a distributed application. These guarantees are embodied in the BRVC abstraction of the FTPP and are explained in greater detail in Sec. V. Once again, the practical impact of this is the reduction of effort required to validate distributed applications software executing on the FTPP.

An often-heard comment is that Byzantine-resilient systems are grossly overdesigned because such strange failure modes cannot occur in real life. On the contrary, we contend that odd unanticipated failure modes occur often enough in practice that their probability of occurrence cannot be assumed away, and that ultrareliable computing systems must be able to tolerate them. We present three examples as evidence.

At least one in-flight failure of a triplex digital computer system was traced to an apparently Byzantine fault and the lack of appropriate architectural safeguards against such faults.[7,11] In circuit-switched network studies at Draper, a failure mode was observed in which a faulty node responded to commands addressed to any node. The garbled response resulted in identification of innocent nodes as failed, until

more sophisticated tests were carried out specifically with this failure mode in mind. A failed processor sending different information to two other processors was observed in the software-implemented fault-tolerance (SIFT) computer.[12] Unless appropriate effort and architectural features are employed to survive, categorize, and analyze failures, Byzantine failures are difficult to identify, making it likely that many other undiagnosed cases of Byzantine failures have occurred.

Because such failure modes exist in practice, an ultrareliable system must be able to tolerate them. Diagnosis of arbitrary failure behavior requires comparison of the device under test with one having at least as many states,[13] implying that component replication is required. The replicated components must be provided with bitwise identical inputs, upon which they perform identical operations. Fault masking, detection, and diagnosis are obtained via bitwise comparison of outputs.

Several key issues arise from the requirement that the system must obtain bitwise consensus on input information such as nonredundant sensor data in the presence of Byzantine failures. Specifically, Byzantine-resilient input-consensus protocols must be implemented. Theoretical studies have resulted in several prerequisites for protocols that correctly function in the face of arbitrary failure behavior by f participants in the protocol. These requirements, appropriate for deterministic and unauthenticated protocols, may be summarized as follows:

1) There must be at least $(3f+1)$ participants in the protocol.[14]

2) Each participant must be connected to each other participant through at least $(2f+1)$ disjoint communication paths.[15]

3) The protocol must consist of a minimum of $(f+1)$ rounds of communication among the participants.[16]

4) The participants must be synchronized to within a known skew of each other.[17]

A system that meets these prerequisites is called "f-Byzantine resilient." In a 1-Byzantine resilient fault masking group (FMG), four participants, each of which is connected to each other participant by three disjoint communication paths, must execute a synchronous two-round protocol to obtain consensus in the presence of a Byzantine fault.

The first and third prerequisites ($3f+1$ participants, $f+1$-round protocol) are readily achieved in a parallel system, but methods to achieve the second ($2f+1$ connectivity) and fourth (synchronization) are not obvious.

IV. Connectivity

In a parallelized or distributed application PE's must communicate with each other for the purposes of the application. Since they must also communicate for the purposes of fault tolerance, the question arises as to how many inter-PE communication paths are necessary to achieve Byzantine resilience. From Ref. 15, we have that as long as there are $(2f+1)$ disjoint communication paths from any participant in the protocols to any other, enough correct information can get through to allow consensus to be reached in the presence of f faults. The intuitive explanation for this connectivity constraint is that the graph representing the set of PE's can be partitioned by a cut set comprising a number of PE's equal to the graph's connectivity. If a majority of these PE's are maliciously faulty, they can collude and cause one partition of the graph to arrive at one consensus value and the other partition to arrive at another.

Because the connectivity prerequisite is absent in most current and proposed parallel computers, one might consider embedding it into their existing interprocessor networks. Unfortunately, the embedding problem is equivalent to the nondeterministic polynomial (NP)-complete subgraph isomorphism problem.[18] Although it may be possible to find an initial embedding by trial and error, finding an embedding in the random graph corresponding to a system suffering failures is difficult and time consuming. Other problems are the efficiency and the diagnosability of the embedding, all of which make an embedding approach unattractive.

Another approach is to group the PE's into completely connected regions or clusters possessing adequate contingent connectivity to allow any PE in a given cluster to be grouped with any others in that cluster to form an FMG. Such clusters could then be interconnected via redundant links into an ensemble of the desired size. However, the physical complexity and number of connectors required to achieve complete interprocessor connectivity within a cluster increases rapidly with the number of PE's in the cluster, quickly leading to the point where the reliability penalty outweighs the reliability enhancement due to reconfigurability.

What is needed is a way to provide the requisite inter-PE connectivity without incurring a severe reliability penalty in the form of an exorbitant number of connectors and wires. To this end, we observe that the PE's of a cluster need not be connected directly in a point-to-point network. Rather, a relatively modest region of theoretically sufficient interprocessor connectivity shared by an aggregate of PE's can adequately service their communication needs. Since the interprocessor connectivity is not typically utilized at all times by a redundant group, it is possible for several redundant groups to time-division multiplex such a region. In this view, interprocessor connectivity is regarded as a valuable resource to be shared among a large number of PE's that subscribe to it through specialized network interfaces.

In addition to the prerequisites previously discussed, a fault-tolerant processor must implement logical and communication functions for fault masking, fault detection, and interprocessor communication. The use of PE's to execute such functions in even a small multiprocessor has been shown to consume excessive PE throughput. For example, the operating system overhead of the implementation discussed in Ref. 19 consumed about 60% of the processors' raw throughput. In the system described in Ref. 20, these functions were performed by dedicated programmable processors called operations controllers. It was estimated[20] that the software execution of these functions by the operations controller was two orders of magnitude too slow for a usable system.

These observations led to the use of specialized hardware components to execute performance-critical fault tolerance-specific functions such as synchronization, interactive consistency, and voting. The FTPP utilizes devices called network elements (NE's) to construct a fully connected, tightly synchronized, Byzantine-resilient hard core for the cluster. The function of the NE is to act as host to several subscriber PE's, for which the NE aggregate performs Byzantine-resilient synchronization, consensus, voting, and consistent ordering functions. The detailed architecture of the NE is given in Refs. 21–23.

Figure 2 shows one possible arrangement of NE's and PE's into a 16-PE, 4-NE cluster. The NE's in the cluster are fully connected to each other via point-to-point communication links, which also serve as physical fault-isolation barriers. These links are used for interprocessor communication and synchronization, and are the only physical connections between primary fault containment regions. Each NE also possesses a port to each subscriber PE. An NE and its associated PE's comprise a primary fault-containment region (FCR). Because of this, a Byzantine-resilient computational group must comprise PE's from disjoint NE's. Figure 2 shows one possible mixed redundancy configuration of the cluster, in which four PE's are grouped into a quadruply redundant processing group denoted Q1, three other PE's are grouped into the triplex group T1, and the remaining PE's are used as nonredundant simplexes S1 through S9. To form a minimum 1-Byzantine resilient FMG, 4 NE's and only 3 PE's are necessary, as explained in Sec. V. As an example of redundancy-management policies possible within this cluster, if a member of quad Q1 fails, then one of simplexes S5, S6, or S7 may be assigned to Q1 to restore its redundancy. Alternatively, Q1

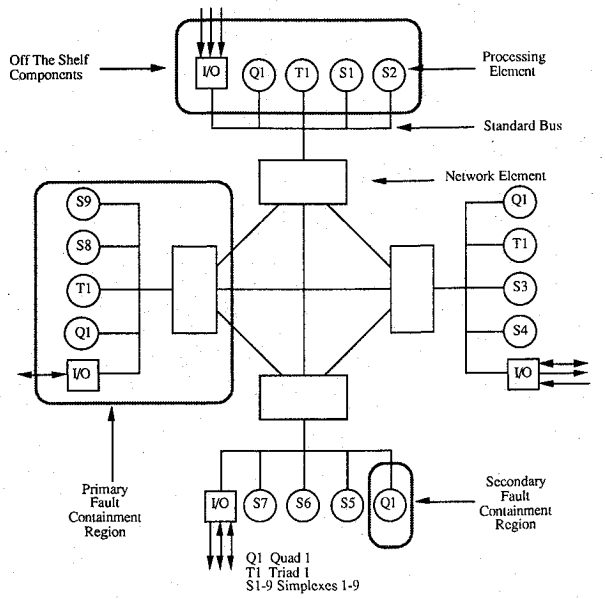


Fig. 2 16-processing element cluster.

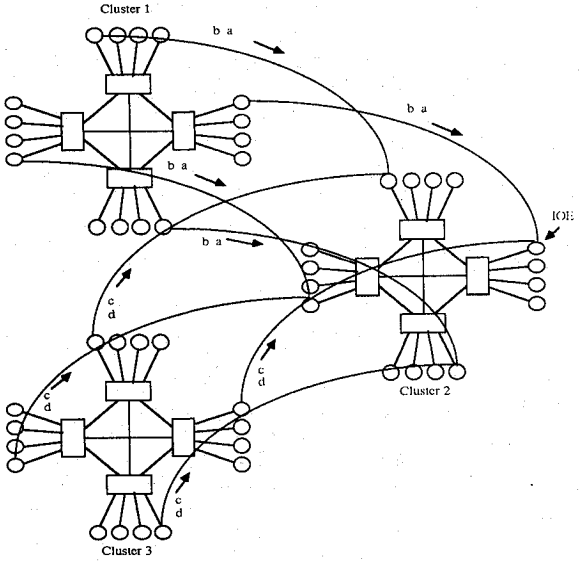


Fig. 3 Intercluster connection using I/O elements.

may be downgraded to a triplex, perhaps named T2. Algorithms required for such real-time reconfiguration have been developed and demonstrated on the prototype FTPP. These algorithms and their performance are described in Refs. 24 and 25.

It is a specific goal of the FTPP architecture to allow such mixed redundancy and parallel-hybrid redundancy management, and the configuration shown in Fig. 2 is but one possible configuration. In addition, the NE interface can be designed to be sufficiently general-purpose that heterogeneous subscribers may be used in the cluster. The CSDL laboratory prototype designs utilize the industry-standard VMEbus as an interface between the PE's and the NE's. Other standardized buses more suitable for flight systems may be incorporated by redesigning the NE bus interface section. As long as they possess the appropriate interface to the NE, some or all of the subscribers may be special-purpose processors, possess large memories, etc. In particular, upgrading an existing system with higher-performance processing elements or input/output (I/O) devices is achieved by replacing the relevant modules, assuming the interface to the NE is kept fixed.

Because of limited NE execution speed and communication link bandwidth, all of the PE's of a large ensemble cannot be

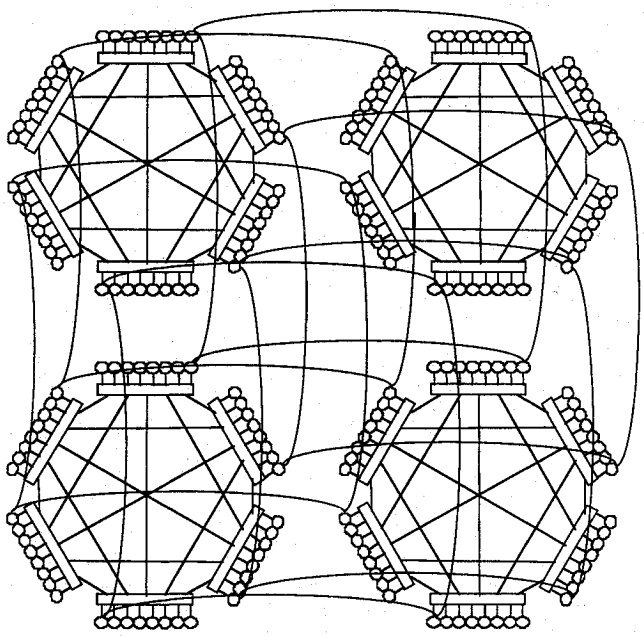


Fig. 4 192-PE fault-tolerant parallel processor.

Table 2 Number of communications ports as a function of architectural approach

Approach	No. of processors	No. of ports
64 simplex processors (hypercube)	64	384
64 quadruplex SIFT-like processors	256	2304
64 triplex FMG FTPP	192	600

efficiently supported by a single cluster. An ensemble is therefore formed from a set of clusters, as shown in Fig. 3. The size of each such cluster can be optimized for some performance parameter using reliability and performance formulations developed in Ref. 22. Given clusters of such an optimal size, specialized I/O elements (IOE's) are used for their interconnection. One IOE subscribes to each NE and its sole function is inter-cluster communication. The IOE's in a cluster are grouped into a redundant IOE group which synchronously executes an intercluster message processing algorithm.

Application-specific input and output device interfaces to sensors, actuators, and communications devices reside on the NE interface bus in the same manner as the PE's and the IOE's. They are controlled by one or more PE's coresident in their FCR. There is no requirement that the I/O device interfaces possess any particular degree of redundancy; their redundancy level is dictated by the criticality of the sensors and actuators with which they interface and the criticality of the computational functions which interface to them. Thus, I/O device interfaces may be simplex, duplex, triplex, or quadruplex in the cluster depicted in Fig. 2. We have found that the use of a standard bus for the PE-NE interface has greatly facilitated the implementation of I/O because such I/O device interfaces can be purchased off-the-shelf. This situation changes somewhat for flight hardware, but it is believed likely that because of the generality of the FTPP's interfaces and the lack of the clock determinism constraint (Sec. V), pre-existing I/O hardware can be used to interface a flight FTPP to the vehicle's sensors and actuators.

For a 64-FMG ensemble, the NE-based approach requires fewer communications ports than an ensemble composed of conventional Byzantine-resilient processors, while the number of ports is 1.58 times that of a 64-PE binary hypercube (Table 2). Figure 4 shows a 64-FMG ensemble (192 PE's) consisting of four clusters each of which has 48 PE's and six NE's for a total of 600 ports.

V. Synchronization

The fourth prerequisite for Byzantine-resilient computation is that members of a redundant group be synchronized to within a known skew. Mutual synchrony is necessary to guarantee termination of any fault-tolerant deterministic distributed decision algorithm,[17] as well as to allow detection of a halted PE. This fundamental problem in fault-tolerant computers is to a large extent responsible for many of the constraints imposed on their design and operation. Moreover, current means for synchronizing fault-tolerant processors and multiprocessors, which include the distribution of high-speed digital clock signals[8] or software-intensive timer synchronization algorithms,[26] do not easily or efficiently extend to synchronizing redundant groups in a potentially loosely coupled, distributed parallel computer. These deficiencies led to a new synchronization scheme for the FTPP called functional synchronization.

For functional synchronization of redundant PE's, all members of a redundant group execute functionally equivalent processes that can be partitioned into segments. The partitioning may be either manual (the programmer uses a "perform synchronizing act" primitive) or automatic (the compiler or operating system appends synchronizing acts onto message transmissions, context switches, reading an input buffer, etc.). Arrival at a segment boundary triggers synchronization of the group by causing it to transmit a message addressed to itself into the NE core, and subsequently await the reception of that message. Because of tight NE synchrony, the difference in time between the delivery of copies of a given message by all nonfaulty NE's is very small. Any PE's awaiting arrival of this message will be synchronized upon its reception.

Members of a redundant group participating in functional synchronization must exhibit an upper-bounded differential execution rate. A degree of nondeterminism and heterogeneity is allowed among the members, but the maximum difference between the execution rates of the fastest and slowest members must be known. In addition, the length of time between synchronizing acts must have an upper bound. A group meeting these requirements possesses an upper bound on the time differential with which different members arrive at a synchronizing act. Members that do not arrive at a synchronizing act within this time bound of a nonfaulty member are faulty by definition. We note that upper bounding the differential execution rate and inter-synchronization time interval is feasible for timer-based iterative real-time control applications.

Functional synchronization avoids the homogeneity constraints inherent in tight synchronization schemes. One benefit of relaxing this constraint is that components composing a redundant group need not be minutely inspected for nondeterministic behavior that might throw members of a group out of synchronization. It is unnecessary to custom-fabricate the ensemble's PE's or the I/O devices they control to ensure that they are clock-deterministic, reducing the difficulty and cost of fabricating the ensemble. Another benefit of removal of the homogeneity constraint is that it is possible to implement hardware and software design diversity in an attempt to overcome common mode faults.

Groups executing functional synchronization consume bandwidth only when necessary, such as when performing synchronization acts, thereby increasing the number of groups that can share the network and reducing the ratio of fault-tolerance-specific hardware to computational hardware. Also, only one network is required to service both the synchronization and communication needs of the processors.

No a priori relationship is imposed among the frames of different groups in the ensemble. Their timing may overlap, be of different lengths and periodicities, and generally exhibit a high degree of disparity. There is no global clock imposed on the system. Synchronization between groups is a natural process of message transmission and reception. Finally, functional synchronization can be transparent to the applications programmer, since acts such as reading input message buffers provide synchronization without explicit action or awareness on the programmer's part.

Two classes of interprocessor message transmissions can be defined based on the redundancy level of the message source. A class 1 transmission emanates from an FMG having at least $(2f+1)$ members. In this class, the redundant messages are transmitted to their destination(s) through disjoint communication paths, followed by a vote of the received copies by each recipient NE. In the face of up to f message source or transmission path faults, $(f+1)$ correct messages will get through and a correct voted copy will be delivered to each nonfaulty recipient.

In a class 2 transmission, the message source is not sufficiently redundant for voting to guarantee that all recipients receive correct or even identical copies. Such a message would be transmitted by a simplex PE in a mixed redundancy system or by a PE interfacing to a nonredundant I/O device to which the other members of the virtual group do not have access. In this case, an $(f+1)$ round Byzantine-resilient interactive consistency exchange is performed to guarantee message validity (the message received by all nonfaulty recipients is identical to that sent) and agreement (all nonfaulty recipients receive identical messages)[14] in the presence of up to f faults. In a two-round, 1-Byzantine-resilient protocol, the first round distributes the source message. The second round distributes copies of the message to all NE's, where the copies are voted and delivered to the PE's. This mixing of class 1 and class 2 transmissions within a theoretically correct Byzantine-resilient structure allows a simple implementation of mixed redundancy.

The network elements provide three basic services to the subscribing PE's: delivery of an identical copy of each message to each of its intended destinations, delivery of message copies at very close to the same time, and delivery of message streams in the same order to each destination. Delivery of identical copies of a message is achieved via the message voting (class 1) process and the interactive consistency (class 2) process. Delivery of copies of a message to different destinations within a small time bound, the property upon which functional synchronization depends, is achieved via tightly synchronous bitwise identical operation of all nonfaulty NE's in a cluster. Delivery of message streams in the same order to each destination is achieved by requiring each nonfaulty network element to synchronously execute the same microcoded message release algorithm on identical input data, specifically, the cluster-wide system exchange request pattern (SERP).

In this message-release algorithm, the NE aggregate sends all enabled messages one at a time. An enabled message is one which passes the valid message condition (at least a majority of the members of a redundant group have requested transmission), the destination free condition (at least a majority of the members of the recipient redundant group do not request flow control), and certain timeout conditions. The NE core synchronously transmits, votes, and delivers a single message at a time. Completion of the message delivery occurs on all NE's at very close to the same time, and messages are delivered in identical order by all nonfaulty NE's. Any mixture of class 1 and class 2 messages may be sent during an NE cycle. When all enabled messages have been delivered, the cycle is repeated with the determination of a new SERP.

Similar to functional synchronization, network elements are synchronized using message transmissions. During a message exchange, whether a class 1 or 2, each NE transmits a message to each other NE. For example, each NE in Fig. 2 would receive three copies of a given message at three receivers, one corresponding to each of its three neighboring NE's. The boundary between two message transmissions from a given neighbor is indicated by temporary cessation of the pulse train shifting data into the recipient NE's message receiver. By comparing the time at which it began a given transmission with its observations of the times at which its neighboring NEs began the corresponding transmission, an NE calculates its skew

relative to them using a fault tolerant comparison algorithm[27] and either advances or retards the time at which it begins the next transmission. This adjustment reduces the skew between the commencement of the subsequent transmission, and keeps the NE's tightly synchronized.

The postsynchronization skew obtained in the prototype FTPP is on the order of ±125 ns. Since this is the same as the skew between delivery of message copies to destination PE's, awaiting the delivery of a given message synchronizes recipients to within this amount. This NE synchronization scheme has been used over both metallic and fiber optic communication media.[21]

It is important to express unambiguously to the users of an architecture the fault-tolerance properties that it embodies. The message passing properties of the FTPP can be concisely expressed in an abstraction called the Byzantine resilient virtual circuit (BRVC) abstraction. This is a guarantee made to the applications programmer on interprocessor message ordering and validity that holds in the presence of Byzantine faults, and relieves the programmer from consideration of faulty behavior when designing a distributed application. An instance of this abstraction is depicted in Fig. 5, which shows two loosely synchronized triplex virtual groups T1 and T2, each of which contains a faulty member. T1 and T2 are sending messages *a, c, x,* and *z* to two other triplexes, T3 and T4, which may also contain faulty members. In cases such as these, the BRVC abstraction provides the following guarantees:

1) Messages sent by nonfaulty members of a source triplex are correctly delivered to the nonfaulty members of recipient triplexes.

2) Nonfaulty members of recipient triplexes receive messages in the order sent by the nonfaulty members of the source triplex.

3) Nonfaulty members of recipient triplexes receive messages in identical order.

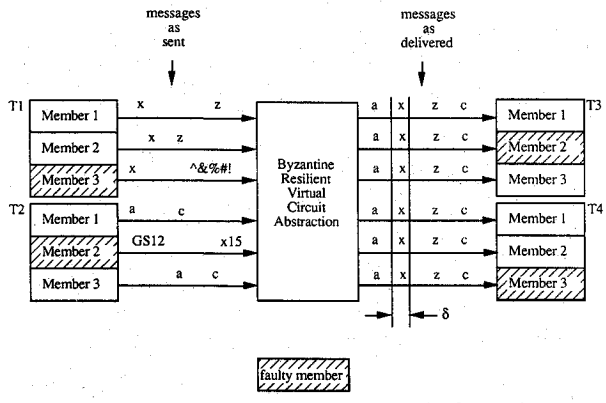


Fig. 5 Byzantine resilient virtual circuit abstraction.

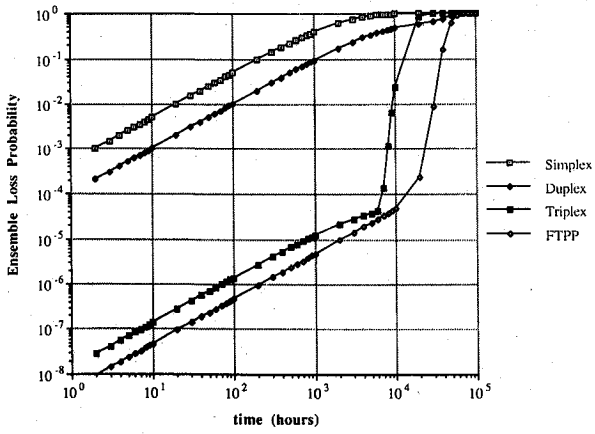


Fig. 6 Ensemble loss probability.

4) The absolute times of arrival of corresponding messages at the members of recipient triplexes differ by a known upper bound (δ in Fig. 5).

These guarantees on totally ordered and timely delivery are not typically made by parallel processors even in the absence of faults. We have found the BRVC to be extremely useful in the construction of robust distributed algorithms on the prototype FTPP.

VI. Results of Reliability Analysis

Using Markov models and combinatorial techniques, the reliability of an FTPP ensemble was computed and compared to the reliability of ensembles using other fault-tolerance techniques. Reliability modeling details are given in Refs. 22 and 28. To fix the problem numerically, it is assumed that the throughput of 64 PE's is required at the beginning of the mission, no repair of permanently failed PE's is performed for the duration of the mission, component failure rates are parameterized according to the number of communication ports possessed by that component, and the PEs or clusters are connected in a binary hypercube. A constant PE failure rate of $\lambda pe = 10^{-4}/hour^4$ and a constant NE failure rate of $\lambda ne = \lambda pe/10$ are also assumed.* The architectures modeled include the following:

1) An ensemble composed of 64 simplex PE's, where each PE possesses no redundancy to aid in fault tolerance but instead depends totally on built-in tests that are assumed to have a fault coverage of 0.95.

2) An ensemble composed of 64 duplex self-checking pairs, for a total of 128 PE's in the ensemble.

3) An ensemble composed of 64 triplicated FMG's, for a total of 192 PE's in the ensemble.

4) An FTPP ensemble composed of 4 NE-based clusters each having six NE's and eight PE's per NE, for a total of 192 PE's and 24 NE's in the ensemble. The PE's are assumed to be operated as triplex FMG's under a parallel-hybrid redundancy-management policy.

Figure 6 shows the system loss probabilities vs mission time for these cases. Figure 7 shows the expected value of the number of processing groups vs time.

The results indicate that, compared to the other approaches modeled, the FTPP approach to providing interprocessor connectivity reduces the short-term system-loss probability, the ensemble's longevity is extended, and the expected value of the number of processing sites is greater. These favorable results are due to the relatively low failure rate of the NE, the reduction of the failure rate of the PE by requiring it to possess only one communication port regardless of ensemble size, and the parallel-hybrid redundancy management scheme made possible by FTPP reconfigurability.

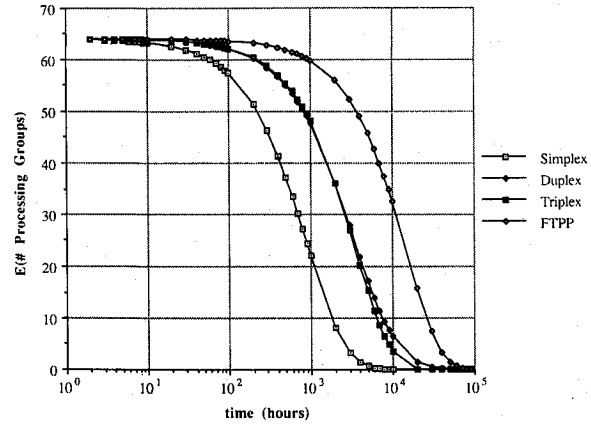


Fig. 7 Expected number of processing groups.

*The transistor count of the network element design is approximately one-tenth that of the processing elements.

VII. Results of Performance Analysis

To quantify the performance of the FTPP, the interval between the time a message is sent by a source group and the time the message is ready to be read by the destination group(s) was computed. Both intracluster messages and intercluster messages may be transmitted by a group. Intracluster messages may be sent to any or all groups in the same cluster as the sender, including the sender itself. Awaiting reception of the latter such message constitutes a synchronizing act in the functional synchronization scheme. Thus, the results of this analysis may be used to compare the performance characteristics of functional synchronization to other fault-tolerant computer synchronization and consistency maintenance methods.

Mean message transmission delay data were obtained using a discrete-event Monte Carlo simulation. A stochastic equilibrium approach[29] was used to determine the traffic density of intercluster messages, which was then used to determine the IOE loading and the intercluster message delay. The mean inter-FMG message-transmission delay was computed as the weighted sum of the intracluster delay and intercluster delay.

Figure 8 shows the results of the simulation for the cases of 4, 8, 16, 32, and 64 triplex FMG's per cluster, corresponding to 16, 8, 4, 2, and 1 clusters per 64-FMG ensemble, respectively. The figure shows the average inter-FMG delay as a function of the mean time between an FMG's message transmissions, or iteration period. Table 3 summarizes the simulation input parameters.

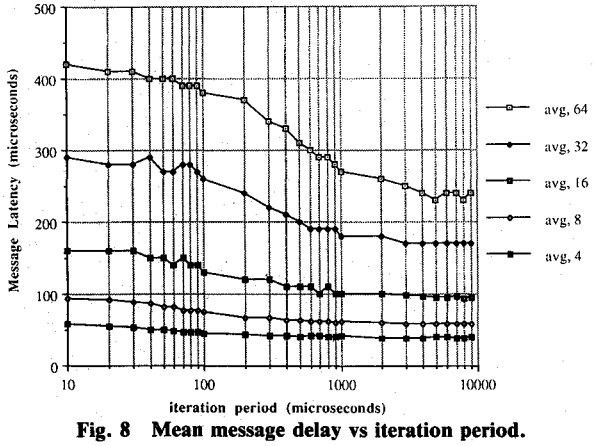


Fig. 8 Mean message delay vs iteration period.

Table 3 Simulation input parameters

Packet size = 32 bytes	64 Mbit/s NE speed
Class 1 transmissions only	Uniform message addressing

The simulation indicates that at an FMG iteration period of around 1 ms, contention for the NE aggregate is reduced to the point that the delay is equal to that incurred when the FMG's message gains immediate access to the network. This low contention limit corresponds to a relatively high iteration rate on the part of the FMG's, indicating that the FTPP can be useful in real-time applications.

Using the simulation, the FTPP's performance was compared to that of two existing fault-tolerant computers. The SIFT computer[19] executed an aircraft control function at a 30-Hz iteration rate, in which the system overhead at tributed to interactive consistency alone was 39%. The FTPP simulation results indicate that the system overhead in the FTPP to obtain interactive consistency, voting, and synchronization ranges from 0.16–0.95%. A second comparison is obtained using a 100-Hz turbine control application.[8] This application presents a relatively taxing set of iteration and data-transfer rates and thus serves as a useful benchmark, consuming roughly 10% of the processor's throughput in the implementation described in Ref. 8. The FTPP simulation results indicate that 0.6–3.5% of FMG throughput is consumed for input-data distribution and validation.

Since message transmission delays are expected to consume under 5% of FMG throughput, an FMG having a raw throughput of 10 MIPS (million instructions per second)† would have a net throughput of 9.5 MIPS with that delay taken into account. Calculation of the effective throughput of a parallel computer depends strongly on the algorithmic structure and how efficiently that structure is mapped onto the hardware. A naive indication of performance is the net throughput T_{net}, simply the number of PE's times the useful throughput of each PE. If the 192-PE FTPP shown in Fig. 4 was configured as 64 triplex FMG's, this quantity would be $T_{net} = 608$ MIPS. The average throughput of the architecture T_{avg}, can be estimated according to an $N/\ell n\, N$ speedup[30] to be $T_{avg} = 146$ MIPS, where N is the number of processing elements. A worst-case estimate T_{worst} based on a $\ell g\, N$ speedup is $T_{worst} = 57$ MIPS. These points are plotted in Fig. 1. It is likely that the actual throughput obtained from a 64-FMG FTPP would fall somewhere between these values, but much more algorithmic development is required before an accurate estimate can be made.

VIII. FTPP Prototypes

Table 4 shows selected characteristics of FTPPs under development at CSDL.

A 16-processor proof-of-concept FTPP prototype, called cluster 1 or C1, has recently been completed to demonstrate the attributes outlined earlier in Table 1. A second cluster C2 was also completed. C1 was built from off-the-shelf single-board-computers that use 12.5 MHz Motorola™ 68020 micro-

Table 4 Selected characteristics of FTPP's

Cluster	No. of PE's/cluster	Total MIPS/ 192-PE FTPP	NE features	Languages
C1 (1989)	16 68020's (12.5 MHz)	115–345 (RAW VAX)	3 CARDS, 64 Mbit/s metal interconnnect	C/Fortran/ADA
C2 (1989)	4 68030's (20 MHz)	184–552 (RAW VAX)	1 card, 64 Mbit/s fiber optic interconnect	C/Fortran/ADA
C3 (1991)	40 68030/80960/ DSP32C's	Depends on PE suite	1 card, 1 ASIC[a] 100 Mbit/s fiber optic interconnect	C/ADA
C4 (1994)	32 GaAs RICS's (100 MHz)	4000–12000 (RAW VAX)	1 card, 4 ASIC's, 1 Gbit/s fiber optic interconnect	C/ADA

[a]ASIC: Application-specific integrated circuit.

†This is the throughput expected to be available from a current technology microprocessor.

Table 5 Fault-tolerance-related data exchange overhead

Event	Microseconds required
PE writes From_A exchange to NE	213
PE writes From_B exchange to NE	213
PE writes From_D exchange to NE	213
PE writes From_D exchange to NE	213
PE reads result of From_A, _B, _C, _D	852
PE writes SYNC exchange to NE	6
PE waits for NE to complete SYNC exchange	10
PE reads result of SYNC exchange	5
PE performs control law calculation	Variable
PE writes From_Vote exchange to NE	201
PE waits for NE to complete From_Vote exchange	50
PE reads result of From_Vote exchange	213
PE writes SYNC exchange to NE	6
PE waits for NE to complete SYNC exchange	10
PE reads result of SYNC exchange	5
PE loops to beginning of control loop	12
Total data exchange time	2222
Total time PE waits for NE	70

processors. The four network elements were custom-designed at CSDL and built from small-scale integration (SSI) and medium-scale integration (MSI) parts. In C1, each network element consists of three 6-U VME cards; in C2 each consists of one 6-U card. In C2 the PE was upgraded to a 20-MHz Motorola 68030 by a straight-forward recompilation of software, thereby showing the easy upgradeability of the FTPP architecture. The inter-NE communication, which was implemented using 64 Mbit/s byte-parallel metallic wires in C1, was replaced by 64 Mbit/s serial fiber optic links in C2.

Performance measurements of the FTPP cluster C1 prototype hardware and unoptimized operating system have been taken and are detailed in Refs. 24 and 25. These measurements were taken in the absence of significant application-specific computational load. They are indicative of the performance of the prototypical unoptimized hardware and operating system, which were designed to demonstrate feasibility of the critical concepts of the FTPP and indicate areas needing further development and optimization. The following quantities are among those measured.

Message Transit Time

The message transit time denotes the time taken for a 40-byte message to be exchanged by the NE aggregate and delivered to the recipient virtual group once it has been written by the sending virtual group. It does not include any operating system formatting, queueing, or other functions, but is instead strictly a measure of the raw NE bandwidth. For a triplex sender, the transit time is 45 μs for a class 1 message and 45 μs for a class 2 message. For a simplex sender, the time taken for a class 2 message is also about 45 μs.

Synchronization Time

The synchronization time denotes the time required for a triplex virtual group to perform a functional synchronization. In the lightly loaded system under test, this time was measured to be 413 μs. It is larger than the message transit time because, unlike the message transit time, the synchronization time includes the operating system overhead required to perform the functional synchronization. Thus, much of the synchronization time can be reduced with increases in processor execution speed and operating system optimization.

Reconfiguration Time

The reconfiguration time refers to the time required to create a triplex from three simplexes (211 ms) and the time required to disband a triplex into three simplexes (570 ms).

Fault Detection, Diagnosis, and Repair Times

The time elapsed between when a member of a triplex virtual group emits an erroneous message and the fault detection and identification function detects the error was measured to be 10 ms. The time required to identify the source of the fault was measured to be 15 ms. The time required to identify a simplex to replace the faulty member, perform the necessary updates to the system configuration, and reinitialize the repaired triplex was measured to be 458 ms.

Performance measurements have also been taken on cluster C2 using a representative control loop application as a benchmark.[21] In the example application, the quadruply redundant controller consisting of channels A, B, C, and D is programmed to iteratively perform a series of four 240-byte class 2 exchanges (From_A, _B, _C, and _D) to achieve interactive consistency on sensor values, a synchronization exchange (SYNC), a dummy computation of a control law, a 240-byte vote of the outputs for delivery to an actuator (From_Vote), and finally another SYNC exchange. The times required to accomplish these tasks are shown in Table 5. Of the 2.2 ms required to perform all fault-tolerant-related exchanges, synchronizations, and voting, 70 μs (3.2%) are spent waiting for the network element to complete a function. This indicates that the 12.5 MHz 68020 processor and the VME bus connecting the processor and the network element used in the test comprise the main limitation on data transfer rate, and that a significantly faster processor and bus can be supported by the network element.

Our objective is to fabricate a flight-ready version of the FTPP (C4) using a gallium-arsenide reduced instruction instruction set computer (GaAs RISC) processing elements and 1 Gbit/s fiber optic interconnects between network elements, with the NE's being implemented in four custom very large-scale integration (VLSI) chips. The raw throughput of six 32-PE clusters would approach 4 billion instructions per second (GIPS) in the all-triplex and 12 GIPS in the all-simplex configuration.

Clusters C1 and C2 have been programmed in C, Ada™, and Fortran. A CSDL-enhanced version of a vendor-supplied Ada™ Run Time System has been ported to C2. This Run Time System supports a 100-Hz pre-emptive rate group scheduler.

IX. Conclusions

A fault-tolerant parallel processor architecture has been developed that combines high throughput and high reliability, with the ability to trade the two in real time. For the architecture to be ultrareliable and validatable, it must be Byzantine resilient, imposing several fundamental constraints on the design, of which meeting the connectivity and synchronization constraints are the major hurdles.

In the fault-tolerant parallel processor, the requisite connectivity is provided to a large number of processing elements through subscription to a smaller number of network elements. Network elements are low-complexity interfaces to a network much smaller than one providing full interprocessor connectivity, but which is sufficient to allow flexibility in the arrangement of the processing elements into redundant groups. Using this feature, redundancy management strategies can defer attrition via parallel-hybrid redundancy, and mixed redundancy is possible within the confines of rigorous Byzantine resilience. In addition, the use of the low-complexity network elements as interfaces to small regions of connectivity reduces processing element complexity, significantly lowering the ensemble's short-term failure rate and increasing its longevity.

Synchronization of redundant processing elements is achieved by functional synchronization. In functional synchronization, redundant processing elements utilize naturally occurring interactions with themselves and the other redundant groups in the ensemble as synchronizing acts. Using this method, members composing redundant groups need not obey the homogeneity constraints imposed by previous synchronous fault-tolerant computers' synchronization schemes. Clock-nondeterministic processing elements may be used as

members of a redundant group, the use of hardware and software design diversity to overcome common mode faults is facilitated, and the integration of heterogeneous processing units and I/O into a theoretically sound architecture is made possible. The redundant groups utilize the shared region of connectivity provided by the network elements only as needed for synchronization, permitting it to be shared by a number of redundant groups, and no additional clock distribution network is required.

A performance model of the fault tolerant parallel processor shows that under a stressful control application less than 5% of a processing element's throughput is spent waiting for completion of intergroup message transmissions. Using this result, the throughput of a fault-tolerant parallel processor containing 192 10-MIPS processing elements grouped into 64 triplex fault masking groups was estimated to lie between 57 and 608 MIPS, depending on the efficiency of the mapping of the parallel algorithm to the architecture.

Two prototype fault-tolerant parallel processor clusters, a 16-processor version and a 4-processor version, have been completed. They include the processing elements, the network elements, the operating system, fault detection, identification and reconfiguration functions, and several demonstration applications. Future plans include extensive evaluation and optimization of the performance of these prototypes and development of a flight-ready system.

Acknowledgments

The authors would like to gratefully acknowledge the financial support of the FTPP's development by CSDL Internal Research and Development funding. They also congratulate the technical contributors to the success of the FTPP project for a job done well. The reviewers are thanked for their constructive comments.

References

[1]Rettberg, R. D., Crowther, W. R., Carvey, P. P., and Tomlinson, R. S., "The Monarch Parallel Processor Hardware Design," *IEEE Computer*, Vol. 23, No. 4, 1990, pp. 18–30.

[2]Hwang, K., "Advanced Parallel Processing with Supercomputer Architectures," *Proceedings of the IEEE*, Vol. 75, No. 10, Oct. 1987, pp. 1348–1379.

[3]Bouricius, W. G., Carter, W. C., Jessep, D. C., Schneider, P.R., and Wadia, A. B., "Reliability Modeling for Fault-Tolerant Computers," *IEEE Transactions on Computers*, Vol. C-20, No. 11, Nov. 1971, pp. 1306–1311.

[4]Dzwonczyk, M., and Stone, H., "A Fault-Tolerant Avionics Suite for An Entry Research Vehicle," *Proceedings of the 8th AIAA/IEEE Digital Avionics Systems Conference*, San Jose, CA, AIAA, Washington, DC, Oct. 1988, pp. 593–598.

[5]Lamport, L., Shostak, R., and Pease, M., "The Byzantine Generals Problem," *ACM Transactions on Programming Languages*, Vol. 4, No. 3, July 1982, pp. 382–401.

[6]Lala, J. H., "Fault Detection, Isolation, and Reconfiguration in the Fault-Tolerant Multiprocessor," *Journal of Guidance, Control, and Dynamics*, Vol. 9, No. 5, 1986, pp. 585–592.

[7]Lala, J. H., "A Byzantine Resilient Fault-Tolerant Computer for Nuclear Power Plant Applications," *Digest of Papers of the 16th Annual International Symposium on Fault-Tolerant Computing Systems*, Vienna, Austria, Inst. of Electrical and Electronics Engineers, Piscataway, NJ, July 1986.

[8]Smith, T. B., "Fault-Tolerant Processor Concepts and Operation," Charles Stark Draper Lab., CSDL-P-1727, May 1983.

[9]Lala, J. H., "Advanced Information Processing System: Fault Detection and Error Handling," *Proceedings of the AIAA Guidance, Navigation and Control Conference*, Snowmass, CO, AIAA, New York, Aug. 1985, pp. 587–596.

[10]Lala, J. H., "An Advanced Information Processing System," *Proceedings of the 6th AIAA/IEEE Digital Avionics Systems Conference*, Baltimore, MD, AIAA, New York, Dec. 1984, pp. 199–210.

[11]Martin, D. L., and Gangsaas, D., "Testing of the YC-14 Flight Control System Software," *Journal of Guidance and Control*, Vol. 1, No. 4, 1978, pp. 242–247.

[12]Palumbo, D. L., Personal communication, NASA Langley Research Center, Hampton, VA, Sept. 1987.

[13]Sundstrom, R. J., "On-Line Diagnosis of Sequential Systems," Ph.D. Thesis, Univ. of Michigan, MI, 1974.

[14]Pease, M., Shostak, R., and Lamport, L., "Reaching Agreement in the Presence of Faults," *Journal of the ACM*, Vol. 27, No. 2, 1980, pp. 228–234.

[15]Dolev, D., "The Byzantine Generals Strike Again," *Journal of Algorithms*, Vol. 3, 1982, pp. 14–30.

[16]Fischer, M. J., and Lynch, N. A., "A Lower Bound for the Time to Assure Interactive Consistency," *Information Processing Letters*, Vol. 14, No. 4, June 1982, pp. 183–186.

[17]Dolev, D., Dwork, C., and Stockmeyer, L., "On the Minimal Synchronism Needed for Distributed Consensus," IBM, Research Rept. RJ 4292 (46990), Yorktown Heights, NY, May 8, 1984.

[18]Tsai, W. H., "Graph Matching Problems: A Survey and Tutorial," *Proceedings of First Conference on Computer Algorithms*, Hsinchu, Taiwan 300, Republic of China, July 1982, pp. 16-1–16-66.

[19]Palumbo, D. L., and Butler, R. W., "A Performance Evaluation of the Software-Implemented Fault-Tolerance Computer," *Journal of Guidance, Control, and Dynamics*, Vol. 9, No. 2, 1986, pp. 175–180.

[20]Walter, C. J., Kieckhafer, R. M., and Finn, A. M., "MAFT: A Multicomputer Architecture for Fault-Tolerance in Real-Time Control Systems," *Proceedings of the IEEE Real-Time Systems Symposium*, Inst. of Electrical and Electronics Engineers, Piscataway, NJ, Dec. 1985.

[21]Abler, T. A., "A Network Element-Based Fault-Tolerant Processor," S.M. Thesis, Massachusetts Inst. of Technology, Cambridge, MA, June 1988.

[22]Harper, R. E., "Critical Issues in Ultra-Reliable Parallel Processing," Ph.D. Thesis, Massachusetts Inst. of Technology, Cambridge, MA, June 1987.

[23]Heyda, R. L., "A Message Passing System for a Fault Tolerant Parallel Processor," S.M. Thesis, Massachusetts Inst. of Technology, Cambridge, MA, June 1987.

[24]Babikyan, C. A., "The Fault-Tolerant Parallel Processor Operating System Concepts and Performance Measurement," Charles Stark Draper Lab., Rept. CSDL-R-2219, Feb. 1990.

[25]Babikyan, C. A., "The Fault-Tolerant Parallel Processor Operating System Concepts and Performance Measurement Overview," *Proceedings of the 9th Digital Avionics Systems Conference*, Inst. of Electrical and Electronics Engineers, Piscataway, NJ, Oct. 1990, pp. 366–371.

[26]Wensley, J., "SIFT: The Design and Analysis of a Fault-Tolerant Computer for Aircraft Control," *Proceedings of the IEEE*, Vol. 66, Oct. 1978, pp. 1240–1255.

[27]Krishna, C. M., Shin, K. G., and Butler, R. W., "Ensuring Fault Tolerance of Phase Locked Clocks," *IEEE Transactions on Computers*, Vol. C-34, No. 8, Aug. 1985, pp. 752–756.

[28]Harper, R. E., "Reliability Analysis of Parallel Processing Systems," *Proceedings of the 8th Digital Avionics Systems Conference*, AIAA, Washington, DC, Oct. 1988, pp. 213–219.

[29]Reed, D. A., and Schwetman, H. D., "Cost-Performance Bounds for Multimicrocomputer Networks," *IEEE Transactions on Computers*, Vol. C-32, No. 1, 1983, pp. 83–95.

[30]Hwang K., and Briggs, F. A., *Computer Architecture and Parallel Processing*, McGraw-Hill, New York, 1984, pp. 27–28.