https://ntrs.nasa.gov/search.jsp?R=19890017288 2020-03-20T02:16:38+00:00Z

NASA Technical Memorandum 4123

Evaluation of Fault-Tolerant Parallel-Processor Architectures Over Long Space Missions

Sally C. Johnson
Langley Research Center
Hampton, Virginia

National Aeronautics and
Space Administration
Office of Management
Scientific and Technical
Information Division

1989

Contents

PRECEDING PAGE BLANK NOT *FILMED*

Introduction

The majority of research in designing ultra-reliable, fault-tolerant systems has been targeted toward a civil air transport application with a 10-hour mission time and perfect maintenance between missions. When considering the design of systems for longer mission times such as a 5-year space mission, it is tempting to try to modify a current architecture by adding additional sparing capability (additional spare components). However, a long mission time can have a surprisingly devastating impact on the reliability of a system. These effects are even more apparent for a parallel-processor architecture because of the multiplicity of system components.

Over a very long mission time, a parallel-processor system with a high number of processors must be expected to incur large numbers of component failures. Several fault tolerance design techniques that are commonplace in today's fault-tolerant architecture designs are unacceptable for highly parallel processors over long mission times. These large numbers of component failures can also have a profound effect on the reliability of a given architecture topology.

The target application investigated in this paper is a Strategic Defense Initiative (SDI) defense satellite requiring 256 processors organized in a parallel manner to provide the required computation throughput. The satellite application analyzed in this study is only one of the hundreds of platforms that would be needed to implement the SDI defense system. A reliability requirement has not yet been set for the SDI, and the number and configuration of platforms is not yet known. For the purposes of this study, the reliability goal assumed is a probability of failure of the total system of less than 0.01 over 5 years, a number often mentioned in connection with the SDI system. Assuming that the platform being analyzed in this study is 1 of 100 satellite platforms, the reliability requirement for a single platform can be approximated as a probability of failure of less than $0.01/100 = 0.0001$ over a 5-year mission. Fully loaded operation of the system during engagement, which will require the 256 parallel computations per platform, will last 30 minutes maximum (from the first missile launch until it no longer matters). The probability of system failure during engagement should be less than 10^{-7} for a single platform.

Survivability of the system under nuclear radiation or other attack conditions will not be considered. Such an attack would probably subject the system to a high number of transient or even permanent faults, significantly affecting the system reliability. However, the characteristics and effects of attacks that the system is likely to encounter are not yet well-understood, and the radiation shielding for the system has not yet been determined.

The environment for the mission will be a space platform. A high rate of transient faults must be expected in the space environment. Transient faults will be assumed to affect only a single component at a time; thus, multiple correlated faults, such as those caused by Sun spots or other radiation, will not be modeled. Only permanent and transient faults will be considered.

Hardware design faults and software faults (sometimes referred to as "generic faults") will not be considered in this study. Unless dissimilar redundant hardware and software components are used, a single design fault can cause system failure by causing simultaneous errors across multiple redundant components. Ricky W. Butler and George B. Finelli of the Langley Research Center have shown that it is infeasible to assure that the probability of system failure due to generic faults is negligible through the use of estimation techniques for a system of any complexity. Validation that the system design is correct through formal proof methods is extremely difficult. However, that topic is outside the scope of this paper.

This paper describes the basic design issues of fault tolerance in the context of the SDI defense satellite application. The fault tolerance potentials of a number of current and proposed fault-tolerant parallel-processor architectures are briefly discussed, and the Fault Tolerant Parallel Processor (FTPP) under development at the Charles Stark Draper Laboratory (CSDL), Inc., is evaluated in some detail. A methodology is then presented for designing a preliminary configuration of a parallel-processor architecture to meet reliability and performance requirements. The methodology is demonstrated by developing and evaluating a preliminary configuration of the FTPP to meet the mission requirements. Finally, several alternative approaches to achieving reliability over long missions are discussed.

Basic Fault Tolerance Features

A number of basic issues of fault tolerance that are vital to the design of a fault-tolerant system are described in this section. A candidate architecture must be evaluated to ensure that it has the potential for meeting the reliability as well as the performance requirements of the mission. If a given architecture has any inherent feature that is contrary to any of these basic fault tolerance requirements, then the architecture must be either altered to provide for fault tolerance or discarded as being inherently not fault tolerant.

Redundancy

An execution of the same code on two or more processors (referred to as "redundancy") is used to detect and correct the effects of faults before they propagate errors into the system. Reliability calculations are based on the assumption that the redundant computations are performed independently; that is, a fault affecting one of the redundant computations must not be able to cause an error in another computation. To enforce such independence, the computations must be performed on separate processors within separate fault-containment regions. Each fault-containment region must have its own independent power supply and must operate independently of other fault-containment regions. Whenever data or system control information is passed between fault-containment regions, voting and fault masking must take place. If the assumption of independence between computations is violated in the actual implementation of the system, then the system built may have considerably lower system reliability than that calculated.

Duplex redundancy is sufficient to detect errors, but triplex redundancy is needed to mask errors. Redundancy is very effective for fault detection and masking if all three computations are performed in exactly the same way on the same input and with the same intermediate system state values. Exact match voting can then be used to detect and mask errors. (In exact match voting, any value that is not bit-for-bit identical to the other values is considered incorrect.) The input values must be exchanged between fault-containment regions and voted to ensure that the same input values are used by all processors. If the processors are not executed in lock-step synchronization and are given differing input values or intermediate system state values because of system asynchrony, exact match voting can no longer be used, and the effectiveness of the fault detection and masking is significantly compromised.

Byzantine Resilience

Unfortunately, components do not always fail in a benign way. Components involved in communication between processors have been known to fail in very "malicious" ways, sending conflicting information to different parts of the system. This type of failure is often very difficult to isolate to the failed component, because if the system does not have enough redundancy to isolate this type of failure properly, a processor receiving the faulty information may appear to be faulty to the rest of the system. A fault-tolerant system that can tolerate any arbitrarily malicious single fault is termed "Byzantine resilient," named for the Byzantine Generals' Problem of generals in the Byzantine army trying to authenticate messages that may be carried by traitorous messengers (Pease et al. 1980).

Malicious failures have been seen in laboratories, and they have even been recorded in flight in which a non-Byzantine-resilient fault-tolerant system was caused to fail (Lala et al. 1986). There is currently no estimate of how likely these malicious failures are. However, we can estimate how unlikely they would have to be for us to be able to ignore them in our system design. To execute the 256 parallel computations of the application with the minimum duplex redundancy would require a parallel-processor system with at least 512 processors. The probability of a single-point failure of each of those 512 processors must be less than 4×10^{-10} per hour to achieve a system probability of failure of less than 10^{-7} over a ½-hour mission. Given a permanent failure rate of 5×10^{-5} per hour and a transient failure rate of 5×10^{-4} per hour, the total rate of failure for a processor is 5.5×10^{-5}. Assuming that malicious failures are the only single-point failures possible, then the probability that each of those failures is malicious must be less than

$$\frac{4 \times 10^{-10} \text{ per hour}}{5.5 \times 10^{-5} \text{ per hour}} = 7.272 \times 10^{-6}$$

Thus, assuming that malicious failures will not happen means assuming that less than 1 out of every 1 375 000 processor failures will be malicious. Additionally, errors made in the design of the system can manifest themselves as malicious failures. Malicious failures may be unlikely, but it is expected that they would not be that unlikely. Therefore, Byzantine resilience should be a requirement for the architecture.

In order for a system to provide Byzantine resilience from any single Byzantine fault, there must be at least four fault-containment regions participating in each message transaction, those message transactions must occur over disjoint communication paths to ensure independence, there must be at least two rounds of data exchange among the fault-containment regions for each message transaction, and the clock skew among the fault-containment regions must be bounded and the bound known. An excellent description of possible causes of Byzantine faults as well as a discussion of how to design a Byzantine-resilient parallel processor may be found in Friend (1986).

The need for Byzantine resilience can have a profound effect on the design of a fault-tolerant parallel-processor architecture. The network topology must be divided into separate fault-containment regions,

into and out of which no errors can propagate. Each computation must be performed in at least two separate fault-containment regions; and whenever data values must pass between regions, they must be compared for fault detection. The communication between the fault-containment regions performing the same computation must take place over distinct communication paths. If a fault-containment region no longer has three distinct communication paths leaving it, then the system is no longer Byzantine resilient. The requirement for redundant computations to take place in separate fault-containment regions can make reconfiguration of complex, connection-limited topologies a difficult problem. The additional requirements of Byzantine resilience significantly compound the difficulty.

Synchronization

If the multiple processors performing a redundant computation are synchronized, then the scheduler can set a maximum allowable time for completion of that computation. This provides a means for the system to determine whether a processor has actually failed or has just been slow to complete a computation.

Failures of the synchronization system must not be able to propagate beyond fault-containment boundaries. Global clocking schemes are usually unsuitable because they present too high a probability of single-point failure of the system. Much research has been conducted on synchronization methods for multiprocessor systems (Pease et al. 1980; Krishna et al. 1985), and a survey of provably correct synchronization techniques can be found in Butler (1988). Various methods range from periodic, loosely coupled synchronization to tightly coupled, lock-step synchronization between processors. Regardless of the method used, the synchronization system is likely to incur a significant overhead to the computational system, and this overhead must be considered when estimating the performance of the system.

Scheduling

A distributed real-time executive must be developed and validated to correctly perform scheduling of tasks in the presence of faults. The scheduling system must be validated to meet the real-time deadlines of the application. The use of data input values in tasks that are produced by other tasks greatly complicates scheduling by adding precedence constraints. The real-time executive must obey the fault tolerance requirements of voting and fault masking between fault-containment regions to maintain data integrity in the presence of faults. Thus, multiple copies of the system scheduler must exist on separate fault-containment regions and must vote on schedules or schedule changes. If the amount of parallelism of the system becomes degraded because of the removal of faulty processors from the configuration, the real-time executive must reassign the tasks among the remaining nonfaulty processors.

Deadlock Protection

A common failure mode of parallel-processing systems is system deadlock caused by excessive parallelism of the computation. The type of deadlock discussed in this section is somewhat different from the type of deadlock usually considered in fault-tolerant processing. Parallel-processing deadlock occurs when every processor in the system is executing a computation that cannot be completed until it spawns a subprocess on another processor. Load balancing of the parallelism is usually accomplished using static parallelism provided before execution, dynamic parallelism controlled by the operating system, or a combination of the two. A discussion of the two methods follows.

Static parallelism is prescribed explicitly by the programmer or a compilerlike tool before execution of the program. If the system provides no deadlock protection or control during execution, then for each application program it must be shown that either system deadlock cannot occur even in the worst-case execution scenario or that the probability of deadlock occurring is sufficiently small. Implementation of a given program in a manner such that it can be proven that its execution cannot lead to system deadlock is at best extremely difficult and usually leads to inefficient use of system resources. The information necessary to determine the probability of system deadlock is usually unavailable before program execution. Similarly, the information necessary to plan for optimal parallelism of the program is also usually unavailable before program execution begins. Therefore, static parallelism alone is usually inefficient and less reliable than dynamic execution or a combination of the two techniques.

A highly reliable parallel-processing system that dynamically controls parallelism must provide a means of system-deadlock avoidance or detection and a recovery that can be shown to reliably complete within an acceptable amount of time. Deadlock recovery schemes are very similar to the schemes used by fault-tolerant systems using fault detection and rollback instead of fault masking. Deadlock recovery usually involves rollback of execution of at least one part of the computation to a previous checkpoint in the program at which the system state was recorded.

Storage of the system state at each checkpoint is expensive in terms of memory and performance. Placing of program checkpoints such that one rollback would always release enough resources to break a deadlock (given any arbitrary dynamic parallelism) would require excessive numbers of checkpoints. The deadline within which rollback and reexecution must be accomplished is dictated by the real-time performance requirements of the system.

There are several methods for handling deadlock avoidance or detection and recovery. Which method to use is an important design decision based on both the system architecture and the nature of the application program. Regardless of which method is used, the probability of deadlock causing system failure must be shown to be acceptably improbable. If deadlock avoidance is used, it must be shown either that the system will always avoid deadlock under all possible circumstances or that the system will avoid deadlock with an acceptably high probability. If detection and rollback are used, the probability that the system will miss a real-time deadline due to deadlock recovery must be shown to be acceptably low.

Adequacy of Communication Network

Another important aspect of the high-level performance analysis is the communication bandwidth between processors. Communication is often a major bottleneck in parallel processing. The system must be considered to have failed if redundant copies of all communications cannot be sent fast enough to satisfy the system performance requirements for the application. Over a long mission time, a large network with high numbers of components will incur large numbers of faults. Serious performance degradation of the communication network in the presence of faults will undoubtedly be one of the most dominant failure modes of a highly parallel processor system.

Preliminary research efforts have shown that current reliability-analysis tools appear to be inadequate for analyzing networks. Reliability analysis of the Integrated Airframe/Propulsion Control System Architecture (IAPSA) candidate architecture using the ASSIST program has resulted in extremely large semi-Markov models because of the combinatoric explosion of states needed to analyze the input/output (I/O) network containing only 36 network nodes (Cohen et al. 1986). Although a number of model reduction techniques, such as model pruning and removal of nondominant failure modes from the model, have been developed for use with the ASSIST and SURE programs (Johnson 1988), an analysis of large networks in the same manner is probably out of the question. A study conducted by the Research Triangle Institute (RTI) under the direction of NASA concluded that the Computer-Aided Reliability Estimator (CARE III) program cannot model large nodal communication networks (Baker and Scheper 1986). New tools or new methods for applying the current tools must be developed.

The most promising approach is to build tools to separately analyze communication networks and interface these tools with current system analysis tools. Such a network analysis tool could be designed to take advantage of the regularity of structure found in most networks to decrease the amount of computation involved. Little research has been conducted in this area, and the networks that have been analyzed have been small I/O networks rather than large, highly parallel communication networks.

An analysis of networks is complicated by the large numbers of components and the resulting combinatoric numbers of component failure combinations. Since a large network contains a large number of components, significant numbers of component failures must be anticipated. In fact, given the reliability of components available today, a major percentage of the components in the network would be expected to fail over a 5–10-year mission. The major validation issues are as follows: (1) ability of the network to correctly reconfigure after a number of failures, and whether such reconfiguration can be accomplished in time to meet real-time deadlines, (2) the number of failures that the network can tolerate before large portions of the system become fragmented, and (3) the number of failures that the network can tolerate before network communication throughput degrades below the performance required for the application.

Network Fault Detection, Isolation, and Reconfiguration (FDIR) algorithms are often rather complex and very difficult to validate. Network FDIR (which typically consists of detecting that one or more failures has occurred, isolating the component or components that are faulty, and determining a suitable new configuration that will isolate any faulty components from propagating further errors through the system) will provide correct and efficient communication and will provide for the detection and tolerance of any later faults that might occur. Validation of an FDIR algorithm includes estimating the probability that the system cannot perform all the above processes before the real-time deadline is exceeded. Failure to reconfigure successfully is likely to happen when numerous failures occur in rapid succession or when multiple failures are difficult to isolate to the failed components. When redundant processors exchange information, it must be assured that

the redundant data communication takes place over independent communication paths, or else the system may be vulnerable to single-point failures.

The probability that the network will become fragmented is a function of both the network topology and the reconfiguration algorithm. Fragmentation occurs when a portion of the system is still operating correctly but cannot contribute significantly to executing the application because communication with the rest of the system is lost or severely degraded. If an insufficient number of links are present in the architecture or if these links are not used efficiently by the reconfiguration algorithm, then fragmentation of the network will be significantly probable over a long mission. Determining the fragmentation probability of a given network would entail determining what fraction of a given number of faults would result in fragmentation of the network. The probability of fragmentation in the presence of each number of faults would then be multiplied by the probability of that number of faults occurring to determine the total probability of network fragmentation.

Degradation of communication performance below the required throughput may occur because of fragmentation, as discussed above, or may simply be the result of large numbers of failures widespread throughout the network. This is likely to be a dominant failure mode of the system if the network topology does not contain enough links or if the system has relatively inflexible reconfiguration. The number of communication links needed to achieve the throughput required for the application can be determined from performance-analysis simulation of the given network configuration. Message contention and routing of communication are a function of the reconfiguration algorithm and the history of locations of the faults that have occurred. With current capabilities there appears to be no way to accurately estimate the overall probability of degraded performance of the network, except for repeating performance simulations for each possible fault-occurrence history of the network.

The state of the art in reliability analysis of communication networks is far below what is needed for SDI parallel-processor applications. Current reliability-analysis techniques are impractical because of the large numbers of components in a large network. Communication-network failures are likely to be a dominant failure mode of these systems. This effect will be shown in later sections when even the rough reliability estimates that are within the state of the art show the need for a very large number of spare communication links to provide a reliable communication network. Therefore, research into the reliability analysis of communication networks should be given a high priority.

State of the Art in Fault-Tolerant Parallel Processors

This section contains a brief analysis of the current state of the art in the development of fault-tolerant parallel processors. The architectures considered range from small-grained parallel processing (in which a single instruction might be assigned to another processor for execution) to large-grained parallel processing (in which processor assignments are at the subroutine level or higher).

Architectures Designed for Ultrareliability

The JPL MAX. The MAX is a high-speed, general-purpose multicomputer designed at the NASA Jet Propulsion Laboratory (JPL) for space applications (Rasmussen et al. 1987). The MAX (named for an Old English word meaning mesh) consists of modules connected by a global dual bus plus a point-to-point mesh network, or meshwork. The MAX operating system, HYPHOS, uses a large-grained data flow scheme of executing functions whenever the data tokens specified on a predetermined execution graph become available. The global bus (a carrier-sense, multiple-access, serial broadcast bus with collision detection) is used for synchronization and function execution assignment. The meshwork, a point-to-point, circuit-switched network, provides the much higher bandwidth needed for transferring data tokens, or even the applications code, between modules. Because of the global bus architecture and centralized control, the designers have stated that the MAX is limited to a maximum of approximately 30 processing modules. Thus, the MAX cannot even approach the performance capabilities needed for the SDI application.

The CSDL FTPP. The Fault Tolerant Parallel Processor (FTPP) was designed by the Charles Stark Draper Laboratory (CSDL), Inc., to provide medium-to-coarse granularity parallel processing for applications requiring high throughput as well as very high reliability (Harper 1987). Out of all the architectures considered, a cluster architecture such as the FTPP appears to be the most likely candidate to provide the reliability and performance needed for the target application. The FTPP is described in detail and evaluated in a later section of this paper.

The AIPS. The Advanced Information Processing System (AIPS) architecture being developed by the CSDL under the direction of the NASA Langley Research Center consists of Fault Tolerant Processors (FTP's) distributed over a mesh network

(Lala 1984; Schabowsky et al. 1984). The AIPS was designed to support loosely coupled distributed processing, not high-performance parallel processing, and its communication bandwidths and processing performance would be unsuitable for the target application (Harper 1987).

Architectures Designed for High Performance

None of the existing high-performance architectures designed without considering high reliability could be modified to have the reliability needed for the SDI application. High reliability is not a feature that can simply be added to an existing architecture. An existing architecture can be made somewhat more reliable by the addition of certain fault tolerance features, such as redundant computation or error-correcting codes. However, the provision of all the necessary fault tolerance features and the elimination of all single-point failures would require a complete redesign of the architecture. Some of the most difficult issues in the design of a fault-tolerant parallel processor are the containment of faults, the assurance that redundant communications always take place over independent paths, the elimination of possible single errors that could significantly degrade the performance of the communication network, and the guarantee of efficient and complete error detection and recovery. Since repair is unavailable in a space satellite environment, a system in which a single failure brings down a large portion of the system would be infeasible for the target application.

Some of the design features commonly used in parallel processors would be difficult to implement in a fault-tolerant manner. Several such features are discussed in the following subsections.

Small-grained parallel architectures. Many architectures obtain maximum parallelism by issuing each instruction or small group of instructions to a different processor for computation. This small-grained parallelism approach would present serious problems for developing a fault-tolerant machine. In order to contain the effects of a fault within one processing node, each communication would require the synchronization and voting of redundant computations. This would seriously impair the high bandwidth communication needed to make the small-grained parallelism efficient. Division of the system into fault-containment regions consisting of a number of processors instead of single processing nodes would help this problem, but this division may be difficult to implement. At the other extreme, the system could be divided into only three or four fault-containment regions; however, the occurrence of only one failure in each region would cause system failure. The system would also have to provide protection from a faulty processor flooding the system with arbitrary messages to arbitrary locations.

Application-specific functional checking. The results of many applications can be checked using application-specific functional checking. For example, Fast Fourier Transform (FFT) computations can be checked by special structures inherent to the computation. The accuracy to which the computations can be checked and the efficiency of the checking algorithm are highly dependent on the application. Additionally, new checking algorithms must be developed for each application. When applicable, functional checking can be extremely efficient by providing protection from any single chip failure with considerably lower hardware overhead. Thus, for applications in which the outputs can be efficiently tested for approximate correctness using functional checking, this approach is a very good one. However, many computations do not lend themselves to functional checking.

Shared-memory architectures. There are some major problems with designing a fault-tolerant parallel processor using a shared-memory architecture. The system must be designed so that no single fault can possibly corrupt shared data. Consider an architecture with duplex-redundant shared memory. If the probability of system failure over the ½-hour engagement mission is to be less than 1×10^{-7}, then the probability of failure of each of the memories would have to be less than 6.5×10^{-4}. The storage/retrieval of correct data in a copy of shared memory can be protected using error-correcting codes (ECC) to detect and correct errors due to transient faults and automatic reconfiguration of memory to avoid using locations corrupted by permanent faults. However, these techniques cannot keep the system from sending corrupt data to be stored in memory. If corrupt data should get into memory by faulty voters, for example, then when those data are subsequently used the system can detect that an error has occurred because the duplex memory contents will not agree when compared. However, the system will not be able to determine which of the two values is the correct one. Thus, very high detection of faults must be provided in the voters. Each time that a data value is to be written into the shared memories, the duplex copies of the data value should be voted for the detection of processor failures. Unless the voters themselves have rigorous self-tests, the data should then be read from the dual redundant shared memories, exchanged, and voted again by another set of voters. This process is time-consuming but necessary to provide detection of faulty voters. The access to shared memory has always been a performance bottleneck of

shared-memory architectures. The additional overhead for two rounds of voting will significantly add to this serious performance bottleneck. The system must also provide enough spare voters to tolerate the large numbers of component failures expected over a long mission and a means for reconfiguration of the voters.

Fixed hypercube architectures. The hypercube architecture, also called the binary k-cube, consists of $N = 2^k$ processors, with point-to-point communication between each processor and its k neighbors. The N processors are assigned unique addresses corresponding to the N, k-bit binary numbers such that any two processors with binary addresses differing by only one digit are neighbors. This addressing scheme allows rapid identification of neighboring processors and facilitates efficient mapping of regular-structured computations onto the hypercube architecture. This mapping scheme depends on every processor in the cube being nonfaulty.

Thus, the features of the hypercube that make it an attractive topology for fast communication and message routing are the same features that make it a reconfiguration nightmare. A plethora of hypercube architectures are under development, and the majority of these systems have fixed communication links between the nodes in the network that make them totally unsuitable as fault-tolerant architectures. Since these fixed links have such an impact on the architecture, hypercube architectures providing dynamic routing will be considered in the next section.

The simplest form of reconfiguration of a hypercube is to remove the dimension of the cube containing a single faulty processor. Thus, a topology of $2^9 = 512$ processors must degrade to a hypercube using only $2^8 = 256$ processors after only a single processor failure. Clearly, it is unacceptable to throw out half the working processors after only one failure.

More complicated strategies consist of providing spare processors in the architecture to replace failed processors. Since rearrangement of the topology in response to a fault precludes the use of the attractive hypercube routing algorithms, the spare must be physically located near the failed processor and must already have all the communication links needed to replace the failed processor in the computation. Several universities, such as the University of Michigan and the University of Illinois, are studying such reconfigurable, fault-tolerant hypercubes. These strategies tend to be extremely complex and require a large number of spare processors spread throughout the configuration to tolerate even a single, localized processor failure. These complex reconfiguration strategies are due to the inflexibility of the hypercube architecture. The analysis of such an architecture is further complicated by the fact that the physical proximity of node failures affects their impact on system reliability. Thus, reliability analysis of a hypercube topology with such a complex reconfiguration scheme is beyond the capability of the tools available and would require lengthy simulation to get even a rough estimate of reliability.

Dynamic hypercube architectures. New hypercube architectures are being developed that can provide both fixed and dynamic routing of messages between nodes. Such dynamic architectures seem to have circumvented some of the problems generally associated with hypercube architectures. Some ideas on designing a fault-tolerant hypercube based on a dynamic routing approach are presented by Rennels (1986). Reconfiguration of a hypercube architecture can be extremely complex, and ensuring that redundant communications take place over independent communication paths can be especially tricky. Although several universities are making progress on some specific issues of fault-tolerant, reconfigurable hypercube architectures, no significant results are available yet. It will be a number of years before the problems associated with designing a highly reliable hypercube architecture are worked out. Such research is proceeding slowly because of the complexity of analyzing a reconfigurable hypercube network topology.

Assessment of State of the Art

The FTPP appears to be the only architecture currently under development that might provide both the high reliability and the high performance required for the target application. Without major modifications, none of the other architectures studied could supply both the high reliability and the high performance needed for the target application. Therefore, the FTPP was chosen as the candidate architecture for further analysis in this paper. The fault tolerance aspects of the FTPP are described in detail in the following section.

Analysis of the CSDL Fault Tolerant Parallel Processor

The Fault Tolerant Parallel Processor (FTPP) was designed by the CSDL to provide medium-to-coarse granularity parallel processing for applications requiring high throughput as well as very high reliability (Harper 1987). The FTPP architecture topology consists of a number of clusters, in which a typical cluster consists of four fully connected Network Elements (NE's), and each NE is connected to four Processing Elements (PE's) as shown in figure 1. The

number of PE's connected to each NE and the number of NE's per cluster can be changed to provide optimal performance for a given application.

Within a cluster, each NE and the PE's connected to it form a fault-containment region. Redundant computations are performed on PE's connected to different NE's, and thus they are in separate fault-containment regions. Communication within a cluster is accomplished by passing the data to the NE's where it is exchanged to provide Byzantine resilience.

Intercluster communication on the FTPP takes place over communication links between PE's of the clusters as shown in figure 2. Three PE's of one cluster must be connected to three PE's of the other cluster over independent communication links to provide Byzantine-resilient data exchange. Each of the PE's should be connected to a different NE so that they are in separate fault-containment regions. An intercluster exchange is requested by the PE's in the first cluster. The NE's in that cluster vote on the data-exchange request. Three PE's on the first cluster then send the data to three PE's on the second cluster. The NE's on the second cluster then vote on the data received. The second cluster then retransmits the voted data to the first cluster where it is again voted for fault detection and isolation.

In the prototype system, each PE is a 12.5-MHz, 32-bit, Motorola MC68020 microprocessor with an MC68881 floating-point coprocessor, 1 megabyte of random access memory (RAM) and 128 kilobytes of erasable programmable read-only memory (EPROM). The FTPP is designed to support various types of PE's, such as I/O processors or other special-purpose processors, in a heterogeneous configuration. The NE's will be custom-built processors, and the communication links are standard buses.

During the design of the FTPP, the system was carefully analyzed and much attention was given to the inclusion of basic fault tolerance features. Therefore, much of the lower-level validation work has already been done. These fault tolerance features are briefly described in the following sections. Since implementation of the FTPP is not yet complete, a number of system parameters such as bandwidth of the NE's are not yet known. Performance analyses will have to be completed to obtain measurements of a number of the parameters used in the reliability analyses.

Redundancy

Duplex redundancy and rollback will be used to detect and correct faults before they propagate errors into the system. The outputs of the two redundant copies of each computation are voted for fault detection. When voting detects an error due to either a

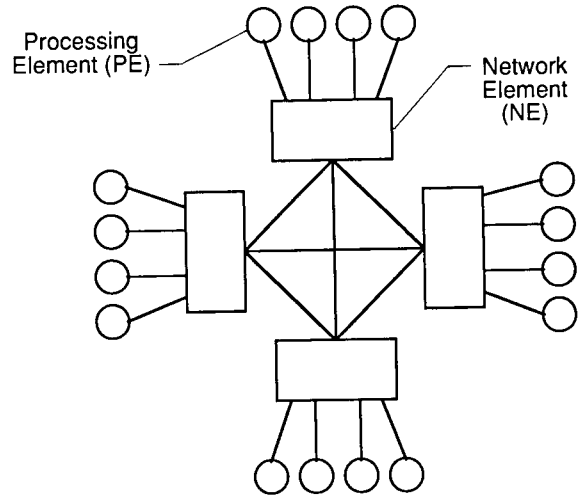

Figure 1. FTPP cluster of 4 Network Elements and 16 Processing Elements.

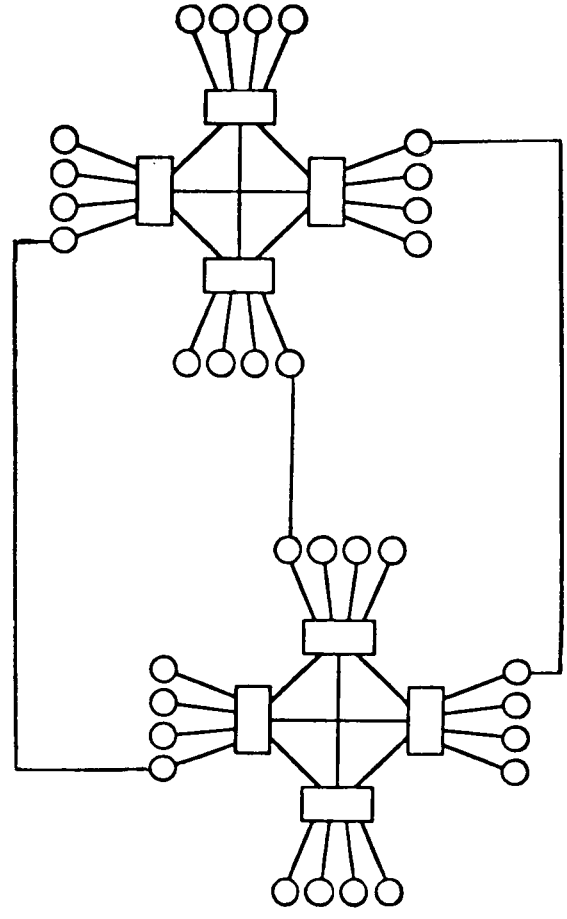

Figure 2. Triplex intercluster communication links between two FTPP clusters.

permanent or a transient fault, the affected computations are repeated from the last checkpoint on one or more independent processors. The three or more resulting independent answers are then all used in the isolation of the cause of the error to a faulty component. A performance analysis of the checkpointing scheme used will have to be performed to determine if the system will always have time to repeat computations from the last checkpoint without missing any real-time deadlines. Because a large number of transient faults are expected, a faulty processor will not be considered to have permanently failed until it produces incorrect answers on several consecutive iterations. Once the processor is determined to have permanently failed, it is removed from the computation and replaced with a spare processor if one is available. If there are no spares, the computations are continued in simplex. System failure due to the arrival of a second near-coincident fault before the first faulty component is removed was not modeled.

The redundant computations must be performed independently so that a fault affecting one of the redundant computations cannot also cause an error in another computation. On the FTPP, the duplex-redundant computations are performed on separate processors within separate fault-containment regions (FCR). Each FCR must have its own independent power supply and must operate independently of other FCR's. Whenever data or system control information is passed between FCR's, voting and fault masking take place.

The effectiveness of redundancy for fault masking also depends on the assumption that the redundant computations are performed in exactly the same way on the same input and with the same intermediate system state values. Exact match voting is then used to detect and mask errors. The input values are exchanged between fault-containment regions and voted to ensure that the same input values are used by all processors.

Byzantine Resilience

Each NE with its associated PE's makes up a separate fault-containment region of the FTPP. The NE's within a cluster are connected to each other via separate point-to-point communication links, thus providing independent communication links between fault-containment regions. Although the computations are only duplex redundant with rollback error recovery, all communications are exchanged between the four or more NE's within the cluster. Thus, all the requirements for Byzantine resilience are met: there are at least four fault-containment regions participating in each message transaction, those messages are sent over distinct, independent communication paths, and two rounds of data exchange occur between the fault-containment regions for each message transaction.

Synchronization

The NE's within a cluster are tightly synchronized. The synchronization algorithm that is used (synchronizing on the second incoming clock signal) has been proven to provide tolerance of one faulty clock for a system of four or more clocks (Krishna et al. 1985). The processors within a cluster of the FTPP remain synchronized within a bounded skew using a functional synchronization scheme as a side effect of the normal intracluster communication. If the PE's do not exchange messages often enough to guarantee synchronization within an acceptable skew, they can initiate data exchanges solely for the purpose of synchronization.

Scheduling and Deadlock Protection

A functional programming model is being developed for the FTPP that will provide graceful degradation in the presence of faulty processors, dynamic load balancing between processors, distributed checkpointing and recovery, and Byzantine resilience with duplex processing (Troxel 1987). In a functional programming model, a function call and its arguments completely describe the computation to be performed. Since no system state is maintained, each function call serves as a global checkpoint for system rollback. If the duplex results of each function evaluation are voted for fault detection, then the errors produced by a faulty processor before detection, and thus the amount of computation to be repeated, are contained within a single-function evaluation (Jagannathan and Ashcroft 1986).

Designing a Preliminary Configuration

Once the candidate architecture is chosen, a preliminary configuration of that architecture must be designed to meet the performance and reliability requirements of the mission. An effective configuration-design methodology for a parallel-processor architecture must combine both performance and reliability evaluation in an iterative approach. Performance analyses range from a high-level determination of the basic requirements of the system to performing trade-off analyses between fault tolerance or performance strategies. Reliability analyses determine design decisions, such as the amount of redundancy needed, and the fault-detection coverages and reconfiguration times required to meet the mission reliability requirements.

The first step is to perform a high-level performance analysis to estimate the number of processors and the number of communication links between processors needed for the application. For this study, it was estimated that 256 processors with at least 1 communication link between them would be needed for computations during system engagement. It is important that these initial estimates be reasonably accurate. However, it is usually impractical to accurately estimate the computation and communication bandwidth requirements of the system before the system is designed in detail and performance analyses can be carried out. If major changes are made later in the required number of processors or the number of communication links, reliability considerations may necessitate a major redesign of the system and many of the analyses performed before the change may have to be repeated. Since the requirements of an application are often considerably underestimated, these initial estimates should be made somewhat pessimistically.

Basic design decisions must also be made in regard to the network topology and on whether duplex-redundancy fault detection and rollback or triplex-redundancy fault masking will be used. Then, a high-level reliability analysis is performed to determine whether the fault tolerance strategy and redundancy levels chosen will provide the required reliability during the ½-hour engagement. Then, the sparing capability and the fault detection and reconfiguration strategies needed to provide the required reliability over the 5-year mission must be determined.

The reliability requirements of the SDI satellite mission require the system to be extremely reliable for a ½-hour maximum engagement mission. The probability that the system cannot perform the engagement mission after 5 years with no repair must be less than 0.0001 for a single platform. In this scenario, the system should have two separate types of FDIR (Fault Detection, Isolation, and Reconfiguration) schemes.

During peacetime when the system is relatively idle, diagnostic tests should be run to ensure that no latent faults are accumulating. Transient faults will probably be more frequent in a radiation-harsh space environment. Thus, the system should make every effort to ensure that faults are not transient before reconfiguring out processors. This is especially important since significant numbers of processors will certainly fail over a 5-year mission, and failed processors cannot be repaired in space. A large number of spare processors will be needed to tolerate the failures expected over 5 years, and these processors should be "cold" (power off when not in use) to minimize their wear-out rates. When a cold spare processor is reconfigured into the system, extensive diagnostics must be performed to ensure that the processor is healthy.

During the short burst when the system is used, the objective is to maximize the throughput of the system. Since the engagement mission time is only ½ hour, a triplex-redundancy scheme with fault masking and minimal reconfiguration capability is preferable to minimize the time spent on fault detection and isolation. A trade-off analysis should be performed to determine if the rollback necessary for fault recovery from duplex-redundancy fault detection is too computationally expensive for the high-performance application. These important design trade-offs must be made by considering both reliability and performance.

Once the basic fault tolerance strategies and redundancy levels are chosen, a high-level reliability model of the system is used to determine if the system has the required reliability during system engagement. If the fault tolerance strategies chosen are found to be inadequate, the system must be redesigned until satisfactory fault tolerance strategies are found. Similarly, if the system is found to provide much more reliability than that needed by the application, the redundancy level or reconfiguration capability can be reduced to lower the cost of the system.

After determining the number of components and connectivity required to meet the reliability requirements for engagement, the next step is to determine the number of cold spare processors needed for a probability of system unavailability of less than 0.0001 at the end of a 5-year mission.

The remainder of the evaluation methodology involves more detailed reliability and performance evaluation. It must be determined if all the basic fault tolerance design issues assumed in the high-level reliability analysis, such as interactive consistency and synchronization, are implemented properly in the system. As the fault tolerance design parameters, such as reconfiguration time, are measured, any affected previous reliability analyses may have to be repeated. If any parameters were previously overestimated to the extent that the system no longer has the required reliability as implemented, the basic fault tolerance strategies or their implementation must be redesigned, and the affected analysis must be repeated.

The Semi-Markov Unreliability Range Evaluator (SURE) reliability analysis program (Butler and White 1988) was used for the analysis of system behavior during the ½-hour engagement mission. The semi-Markov models for SURE to analyze were generated using the Abstract Semi-Markov Specification

Interface to the SURE Tool (ASSIST) computer program (Johnson 1986).

The 5-year mission analysis required the solution of models with extremely long path lengths over a long mission time. None of the well-known reliability analysis programs developed by NASA Langley—the SURE program, the Computer-Aided Reliability Estimator (CARE III) program (Bavuso and Peterson 1985), and the Hybrid Automated Reliability Predictor (HARP) program (Dugan et al. 1986)—were suited to solving these models. Neither CARE III nor HARP could generate the models needed for the long-mission analyses without extensive precomputation, because they cannot model cold spares. The HARP program also allows the user to input a hand-generated model; however, the HARP solver could not handle long-path-length models with as few as 50 transitions over a 5-year mission time. The SURE program required computation time on the order of several days to solve a single model.

Therefore, the little-known Scaled Taylor Exponential Matrix (STEM) program (Butler and Stevenson 1988) was used to solve the long-mission-time models. The STEM program, which can solve only Markov models, has the same model-input format as the SURE program. For most large models, the SURE program is considerably faster than the STEM program. However, the STEM program can solve simple models with very long path lengths and long mission times in seconds that would require days of execution time for the SURE program. The ASSIST program was used to automatically generate the models to input to STEM.

Reliability of Individual Components

The first step in performing a reliability analysis of a system is to estimate the failure rates of the individual components making up the system. Although exact determination of component failure rates is impossible before the components are accurately characterized, rough estimates of component failure rates must be made in order to perform trade-off studies.

The presence of an abnormality in a component that will cause it to operate incorrectly is referred to as a "fault." A component containing such a fault is referred to as "having failed." If the failed component is subsequently used in computation, it may introduce an error into the computation.

Calculations of component failure rates according to the 1982 Military Handbook 217D (MIL-HDBK-217D) take the harshness of the environment into account by rating the class of environment in which the component will operate and then by multiplying the failure rate of the component by the corresponding environmental factor. The environment of a satellite is under the class "Space, Flight" environment for Earth orbital vehicles that are neither under powered flight nor in atmospheric reentry. The probability of failure for a component in this environment must be multiplied by 0.9. This environment is considered only slightly harsher than the "Ground, Benign" environment, which has an environmental factor of 0.38. During deployment of the satellite, the components will undergo a period of extremely harsh environment under the "Missile, Launch" environment class, which has an environmental factor of 13.0. Thus, a significant number of components may be lost during the initial deployment of the satellite.

The 1982 Military Handbook 217D standard also includes the effects of the use of new devices or technology in the failure-rate calculations. Current technology parts will be assumed for this study because it takes 5–10 years for technological advances to be available in space-qualified parts. Even with current technology, the devices used in a highly parallel processor for an SDI satellite would almost assuredly be new devices. Up to the first 6 months of production of a new device, the failure rate of the device must be multiplied by a factor of 10.0 to account for the learning factor. Also, if Silicon on Sapphire (SOS) technology were used to counter the high-radiation environment, this would be another reason for using a learning factor of 10.0 according to the military standard since the use of SOS technology is explicitly listed as requiring a learning factor of 10.0.

The failure rate of a Bendix BDX-930 avionic processor was measured as 4×10^{-4} per hour, of which 1.4×10^{-4} per hour was due to connector failures (McGough and Swern 1981). However, as Very Large Scale Integration (VLSI) technology improves, marked improvement in the reliabilities of simple components can be expected. The following component probabilities of failure will be used. It will be assumed that extremely simple VLSI chips can be produced with a probability of failure of approximately 10^{-7} per hour. More complex chips, such as switches and I/O nodes with very little computational capability, are assumed to have probabilities of failure as low as 10^{-6} per hour. Physical connectors will also be assumed to have probabilities of failure of 10^{-6} per hour. A probability of failure for a computational processor of 5×10^{-5} per hour will be assumed. Computational processors tend to be extremely complex and need separate power supplies and internal memories, etc. Thus, since the failure of a computation processor can be caused by the failure of any one of these components, the failure rates of the individual components add up to make the processor failure rate very high. The additive nature of these component failure rates can be seen in the

reliability analysis done for the entry research vehicle (ERV) by Dzwonczyk et al. (1989). This study also shows the dramatic increase of component failure rates in the presence of heat and other environmental hazards of space.

The failure rate of computational processors includes permanent failures, transient failures that persist long enough to be detected as transient, and transient faults that cause permanent errors (such as corrupted memory locations). Intermittent faults will not be modeled in this study because so little data are available on their arrival and disappearance rates that they cannot be modeled with reasonable accuracy.

It has been suggested by some that processor failure rates over long mission times may not be exponentially distributed, but may actually decrease over a long mission time (Hecht and Hecht 1985). A brief description of the applicability of this phenomenon to the target application and a reliability analysis taking this into account are given in the subsequent section entitled "A More Optimistic Long-Mission Analysis."

To provide redundancy over the long mission, the system will use spare processors that are cold (power off when not in use). Cold spare processors are subject to failure even when the power is off; however, it is generally assumed that they have a lower failure rate than "hot" (power on) processors. The failure rate of cold spare processors is a subject of debate among reliability engineers, mostly because of the lack of available data. Most failure-rate estimates range from one-third to one-twentieth (see Kern 1978) those of active processors. For this study, cold spare processors will be assumed to have failure rates that are one-tenth those of active processors. It will be assumed that the cold processors are extensively tested for failures before being brought into the active configuration. The use of spare processors that are "warm" (periodically cycled on and checked for latent faults) will not be considered because the act of repeatedly turning these processors off and on actually causes them to have a significantly higher failure rate.

For this study, active-component failures rates for the FTPP were assumed to be 5×10^{-5} per hour for the PE's and 1×10^{-5} per hour for the NE's. Since the NE's can be considerably simpler and have less memory than the PE's, they have a correspondingly lower failure rate. Communication links are assumed to have failure rates of 1×10^{-6} per hour because of their physical connectors.

Designing a Preliminary Configuration for the FTPP

In this section, a preliminary configuration is designed for the FTPP to meet the requirements of the application. Those requirements are that the probability of failure of the system to provide 256 parallel, reliable computations during the ½-hour engagement must be less than 10^{-7}, and the probability that the system will not have sufficient resources to support the reliable engagement after 5 years must be less than 0.0001.

Since the FTPP provides Byzantine-resilient fault tolerance using only duplex redundancy, a fault tolerance strategy of duplex redundancy with no reconfiguration during engagement was chosen for the first cut. This requires 512 processors. Two choices that must be made to optimize performance and reliability for the application are how many PE's should be connected to each NE and how many NE's should there be per cluster. A number of interrelated factors go into making this decision. Each duplex computation must take place on two PE's in the same cluster that are connected to different NE's. Communication within a cluster (intracluster communication) is very fast, and every communication is exchanged between all the NE's in the cluster. Communication between clusters (intercluster communication) takes place over links between PE's on each of the clusters. This is much slower than intracluster communication, and if too much intercluster communication is required, then the PE's involved in the communication will be less able to contribute to the application computation. Thus, the divisibility of the application processing into somewhat independent clusters is mostly a function of how well the application program computation can be broken into somewhat independent pieces.

The choice of system configuration should be based on a combination of performance analysis and reliability analysis of the system. For the purposes of this analysis, it will be assumed that reconfiguration can take place only within a cluster, and each cluster must have its own pool of spare processors. This assumption has three effects on the system. First, a lower reconfiguration time can be assumed because reconfiguration will not have to take place systemwide. Second, the communication performance requirements between clusters can be lower because the high bandwidth needed for timely function migration (transmitting code to a new host processor) is not needed. Third, this assumption of no function migration will lower the memory requirements for each processor because the processors

within each cluster will have to keep in memory only the code that will be processed within the cluster.

Since PE and NE failures within a cluster happen independently from PE and NE failures in other clusters, each cluster can be analyzed independently when determining the sparing capability needed within each cluster. If the system is broken into 16 clusters of 32 parallel computations each, the probability of failure of each of the 16 clusters over ½ hour must be

$$10^{-7}/16 = 6.25 \times 10^{-9}$$

Since this is less than the failure rate of a single NE, the cluster must have enough sparing capability to tolerate the failure of any arbitrary NE, as well as the failure of several PE's. Thus, if each NE supports only two PE's, then a single NE failure causes only the loss of two PE's; whereas if there are five PE's per NE, a single NE failure causes the loss of five PE's. Therefore, the more PEs per NE, the more spare processors are needed in the configuration. The above analysis did not take into account the fact that each additional NE in the cluster requires links to each other NE in the cluster. Thus, the number of point-to-point communication links between NE's in a cluster is

$$\#\text{LINKS} = \#\text{NE} \times (\#\text{NE} - 1)/2$$

Thus, the number of intracluster communication links grows from 21 links required to fully connect 7 NE's to 136 links required to connect 17 NE's. To support 32 parallel computations, the system with 2 PE's per NE requires the participation of 17 NE's in every intracluster communication exchange, whereas the system with 5 PE's per NE requires only 8 NE's to participate in intracluster communication exchange. This additional message exchange overhead can seriously degrade performance. According to the system designers, a cluster with more than 8 or 10 NE's is probably infeasible. Unless the application can be split into a very large number of clusters, it is probably preferable to have as many PE's per NE as the communication throughput of the NE can reliably handle. The divisibility of the application processing into somewhat independent clusters, coupled with a performance analysis of the communication throughput that can be handled by an NE, should be the driving factors in choosing the number of PE's per NE and the number of NE's per cluster.

As evidenced by the above discussions, choosing the optimum configuration involves a complex trade-off analysis. The more PE's there are per NE, the fewer NE's there are in the system to fail—but more spare PE's are needed. The more NE's there are in a cluster, the less intercluster communication is needed—but more NE's are involved in intracluster exchanges. Robert Burley, manager for business and systems engineering of advanced VAX systems at Digital Equipment Corporation, has a good analogy: "It's like a game called Whack-A-Mole. You stand there with your mallet and when a mole pops up, you hit it on the head. But the faster you hit, the faster other moles pop up. The next mole up can be CPU speed, memory bandwidth, I/O port performance, or device speed. You just have to keep hitting." (See Wilson 1987.)

For our first attempt, the initial configuration will consist of 16 clusters with each cluster responsible for 32 redundant computations. Thus, system failure occurs unless all 16 clusters still have at least 32 PE's working at the end of the engagement. These 32 duplex computations will require 64 parallel computations, which is probably approaching the limits of what a single cluster can handle. To obtain an overall probability of failure of less than 10^{-7} over a ½-hour engagement, each of the 16 clusters must have a probability of failure of less than $10^{-7}/16$, or 6.25×10^{-9}. Table 1 shows the probabilities of failure over ½ hour determined for various cluster configurations as computed by the SURE reliability analysis program. The failure rates of PE's and NE's are high enough that the cluster must have enough sparing capability to tolerate failure of at least one NE and several PE's during engagement. Details of the reliability models used in the above analysis are given in appendix A.

For the purposes of this paper, it will be assumed that a performance analysis showed that each NE can provide the throughput needed for six PE's. A cluster consisting of 6 NE's with 6 PE's attached to each plus 1 NE with 2 PE's attached to it for a total of 38 PE's, as shown in figure 3, will be used in further analysis in this paper. This configuration has been shown to provide the needed reliability and minimizes the number of NE's in the cluster without overloading the communication and voting that each NE must perform. Clusters formed from the various possible configurations of 38 PE's attached to 6 NE's do not differ significantly in reliability.

Reliability Over a 5-Year Mission

The configuration chosen for the ½-hour mission consists of 16 clusters with each cluster containing 7 NE's and 38 PE's and a maximum of 6 PE's per NE. This requirement was represented in table 1 by the configuration of six NE's with six PE's each plus one NE with two PE's. However, any cluster

Table 1. Failure Probabilities Over ½ Hour for Each of 16 Clusters[a]

Maximum PE's per NE	Configuration	Total PE's	Cluster failure probability	Sufficiently low failure probability
2	16×2	32	2.7627×10^{-6}	No
2	$16 \times 2 + 1 \times 1$	33	5.0317×10^{-7}	No
2	**17×2**	**34**	**3.6841×10^{-10}**	**Yes**
3	11×3	33	5.4222×10^{-7}	No
3	$11 \times 3 + 1 \times 1$	34	2.9302×10^{-7}	No
3	**$11 \times 3 + 1 \times 2$**	**35**	**1.9700×10^{-10}**	**Yes**
4	8×4	32	2.4677×10^{-6}	No
4	$8 \times 4 + 1 \times 1$	33	5.4503×10^{-7}	No
4	$8 \times 4 + 1 \times 2$	34	3.7563×10^{-7}	No
4	$8 \times 4 + 1 \times 3$	35	1.9434×10^{-7}	No
4	**9×4**	**36**	**1.4220×10^{-10}**	**Yes**
5	7×5	35	3.1277×10^{-7}	No
5	$7 \times 5 + 1 \times 1$	36	1.6982×10^{-7}	No
5	**$7 \times 5 + 1 \times 2$**	**37**	**1.1115×10^{-10}**	**Yes**
6	6×6	36	2.6775×10^{-7}	No
6	$6 \times 6 + 1 \times 1$	37	1.4537×10^{-7}	No
6	**$6 \times 6 + 1 \times 2$**	**38**	**9.4635×10^{-11}**	**Yes**
7	5×7	35	4.1573×10^{-7}	No
7	$5 \times 7 + 1 \times 1$	36	3.3892×10^{-7}	No
7	$5 \times 7 + 1 \times 2$	37	2.3383×10^{-7}	No
7	$5 \times 7 + 1 \times 3$	38	1.2098×10^{-7}	No
7	**$5 \times 7 + 1 \times 4$**	**39**	**7.9106×10^{-11}**	**Yes**
8	4×8	32	2.3603×10^{-6}	No
8	$4 \times 8 + 1 \times 7$	39	9.6674×10^{-8}	No
8	**5×8**	**40**	**7.6777×10^{-11}**	**Yes**

[a]A configuration description of $5 \times 7 + 1 \times 2$ denotes a cluster containing five NE's with seven PE's attached to them plus one NE with two PE's. The probability of failure over the short-term mission for each of the 16 clusters must be

$$10^{-7}/16 = 6.25 \times 10^{-9}$$

Note that the probability of failure drops dramatically for configurations containing enough spares to tolerate any arbitrary NE failure.

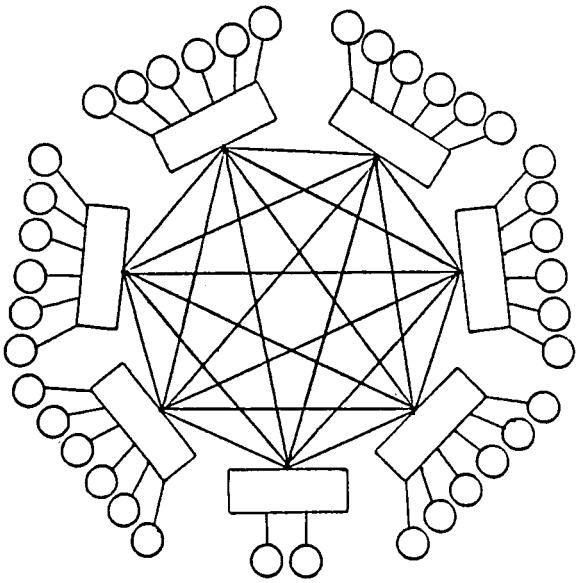

Figure 3. A cluster consisting of 6 NE's with 6 PE's attached to each plus 1 NE with 2 PE's attached to it for a total of 38 PE's.

configuration with 7 NE's and 38 PE's with no more than 6 PE's connected to 1 NE meets the reliability requirements of the ½-hour mission. The choice of a configuration without a full complement of PE's on each NE may be confusing to the reader; however, it should be remembered that this is the target minimum configuration needed at the end of a 5-year mission after a number of PE and NE failures have occurred, not the initial configuration to be launched at the start of the mission.

In this step it is determined what the initial configuration of the system must be to provide the sparing capability to still have 38 PE's working in every cluster after 5 years with a probability of 0.9999. Since PE's cannot be shared across clusters, each cluster must provide for its own sparing. Since the system has been divided into 16 independent clusters, the probability of having fewer than 38 PE's working after 5 years for each of the 16 clusters must be less than

$$0.0001/16 = 6.25 \times 10^{-6}$$

An analysis of various configurations of 38 PE's with up to 6 PE's per NE that were tested over a ½-hour mission shows that as long as there are 38 PE's per cluster in the configuration, the probability of failure is the same or lower. For example, a cluster with nine NE's with four PE's per NE plus one NE with two PE's has a probability of failure of 1×10^{-11}, which is two orders of magnitude lower than the chosen configuration. As the average number of PE's per NE drops, the probability of failure drops by several orders of magnitude. Thus, what is important is that 38 working PE's are connected to some number of working NE's in the cluster.

In order to provide reliability over the long mission, cold spare PE's connected to cold NE's will be used. As discussed earlier, these components will be assumed to have one-tenth the failure rate of hot PE's and NE's. When a cold spare is brought on-line to replace a failed PE, the processor must be fully tested for latent failures that may have occurred while the processor was dormant. Since this process takes a considerable amount of time, the system must always have at least the 38 processors needed for the active mission running hot (power on) at all times. As the hot processors fail, they will be replaced from the pool of cold spares. It is possible that the system can enter the engagement mission while one or more clusters is in the process of reconfiguring in a cold spare processor, and thus the system would have less than a full complement of 38 PE's. However, for purposes of simplification, this possibility will not be modeled because of the negligible probability of an occurrence and its minimal effects on the rough reliability estimates.

A simple combinatoric calculation cannot be used to determine the initial system configuration needed because of the reconfigurations and differences in failure rates. Therefore, a Markov analysis of the system is required. The Markov analysis method consists of choosing a candidate initial configuration, performing a Markov analysis, and checking the results to see if the needed reliability was obtained. If the reliability is too low, the number of initial PE's per cluster is increased, and the process is repeated until an optimum initial configuration is reached.

To get an idea of the number of spares needed, a simple model was used that included only PE failures. The ASSIST file to generate this model is shown in appendix B. This analysis showed that even if the NE's do not fail at all, an initial configuration of 31 NE's with 6 PE's each is needed for a total of 186 PE's per cluster. Of those 186 PE's in a cluster, 148 are cold spares, 6 are hot spares, and 32 are needed for performing the 16 computations with duplex redundancy. This number is clearly optimistic because NE failures were not considered, and each NE failure will additionally take out all the PE's connected to it. This rough approximation gives us a starting point for the larger, more accurate models of failure behavior.

Unless great care is exercised in modeling the system, exorbitantly huge models will be obtained that cannot even be stored on our computer, much less solved. Consider, for example, our current configuration of 31 NE's and 186 PE's. At least

31 − 6 = 25 of the NE's or 186 − 38 = 148 of the PE's, or some combination of NE's and PE's must fail before we fail to have the needed configuration. There are 186!/38! distinct paths to system failure from PE failures alone. Fortunately, many of these paths can be aggregated and many others are unlikely enough that they can be ignored.

Before proceeding further, we should consider the feasibility of a cluster of 31 NE's. The point-to-point communication links to fully interconnect these 31 NE's would require $31 \times 30/2 = 465$ links. Getting close enough to these 31 NE's and their 186 PE's to provide tight synchronization and small intracluster message latency is probably impossible. One solution is to try to break up the system into much smaller clusters. If our application algorithm cannot be broken into many somewhat independent pieces, then we clearly have a problem. In order to reduce the size of each cluster, the system was divided into 32 and then 64 clusters and the above analyses were repeated.

If the system is divided into 32 clusters, each of the clusters is responsible for the reliable execution of 8 duplex parallel computations. The probability of failure of each of the clusters over the ½-hour engagement mission must be less than

$$1 \times 10^{-7}/32 = 3.125 \times 10^{-9}$$

Table 2 shows the ½-hour failure probabilities for each of the 32 clusters. Over the 5-year mission, the probability of falling below the number of processors needed for engagement must be less than

$$0.0001/32 = 3.125 \times 10^{-6}$$

Using a configuration of 6 PE's per NE requires an initial configuration of approximately 21 NE's and 126 PE's. This initial configuration would, of course, require $21 \times 20/2 = 210$ point-to-point communication links between the 21 NE's. Alternatively, using a configuration of only 4 PE's per NE requires an initial configuration of 28 NE's and 112 PE's. If a cluster of 21 NE's is infeasible because of interconnection and physical-proximity restrictions, the system must be divided into even more clusters.

With the system divided into 64 clusters, each of the clusters is responsible for the reliable execution of only 4 duplex parallel computations. The probability of failure of each of the clusters over the ½-hour engagement mission must be less than

$$1 \times 10^{-7}/64 = 1.5625 \times 10^{-9}$$

Table 3 shows the ½-hour failure probabilities for each of the 64 clusters. Over the 5-year mission, the probability of falling below the number of processors needed for engagement must be less than

$$0.0001/64 = 1.5625 \times 10^{-6}$$

Using a configuration of 6 PE's per NE requires an initial configuration of approximately 15 NE's and 90 PE's per cluster. Each cluster would require $15 \times 14/2 = 105$ point-to-point communication links between the 15 NE's. Using a configuration of only 4 PE's per NE requires an initial configuration of 19 NE's and 76 PE's per cluster. If a cluster of 15 NE's is infeasible because of interconnection and physical-proximity restrictions, the system must be further divided. However, dividing the computation into more than 64 clusters (with each cluster responsible for fewer than 4 duplex parallel computations) with the requirement that very little data pass between clusters is expecting too much from even the most optimally parallelizable applications.

With the system divided into 64 clusters of 15 NE's and 90 PE's each, the total number of components in the system so far is $64 \times 15 = 960$ NE's, $64 \times 90 = 5760$ PE's, plus $64 \times 105 = 6720$ intracluster links. The technical difficulties of putting that much hardware on a satellite and launching it into space will not be considered in this paper. In the following section, the number of links needed to provide communication between these 64 clusters will be determined.

Intercluster Communication

One problem that has been ignored up until now is how many intercluster communication links will be needed. Since the larger clusters are probably infeasible because of their interconnection, communication bandwidth, and tight synchronization requirements, only the intercluster links needed for the 64-cluster system will be considered in this section.

For the purposes of this analysis, it has been assumed that the number of intercluster communication exchanges needed will be small enough that the overhead will not seriously degrade the computational performance of the PE's involved. The amount of intercluster communication needed is dependent on the application being computed. However, for a system divided into 64 clusters with each cluster responsible for performing only 4 parallel computations, this is probably not a valid assumption. If this assumption does not hold, then more PE's must be added to the configuration to handle this communication overhead processing, and still more spares will be needed over the 5-year mission.

The ASSIST file shown in appendix C was used to to get a rough estimate of the number of links needed

Table 2. Failure Probabilities Over ½ Hour for Each of 32 Clusters[a]

Maximum PE's per NE	Configuration	Total PE's	Cluster failure probability	Sufficiently low failure probability
2	8×2	16	5.8932×10^{-7}	No
2	$8 \times 2 + 1 \times 1$	17	8.5944×10^{-8}	No
2	$\mathbf{9 \times 2}$	**18**	$\mathbf{3.1091 \times 10^{-11}}$	**Yes**
3	6×3	18	5.7490×10^{-8}	No
3	$\mathbf{6 \times 3 + 1 \times 1}$	**19**	$\mathbf{1.9438 \times 10^{-11}}$	**Yes**
4	4×4	16	5.4849×10^{-7}	No
4	$4 \times 4 + 1 \times 3$	19	3.8248×10^{-8}	No
4	$\mathbf{5 \times 4}$	**20**	$\mathbf{1.4838 \times 10^{-11}}$	**Yes**
5	4×5	20	3.8163×10^{-8}	No
5	$\mathbf{4 \times 5 + 1 \times 1}$	**21**	$\mathbf{1.2661 \times 10^{-11}}$	**Yes**
6	$3 \times 6 + 1 \times 3$	21	3.0224×10^{-8}	No
6	$\mathbf{3 \times 6 + 1 \times 4}$	**22**	$\mathbf{9.6138 \times 10^{-12}}$	**Yes**
7	$3 \times 7 + 1 \times 1$	22	3.1826×10^{-8}	No
7	$\mathbf{3 \times 7 + 1 \times 2}$	**23**	$\mathbf{9.9624 \times 10^{-12}}$	**Yes**
8	$2 \times 8 + 1 \times 7$	23	2.1569×10^{-8}	No
8	$\mathbf{3 \times 8}$	**24**	$\mathbf{9.7905 \times 10^{-12}}$	**Yes**

[a]The probability of failure over the short-term mission for each of the 32 clusters must be

$$10^{-7}/32 = 3.125 \times 10^{-9}$$

Table 3. Failure Probabilities Over ½ Hour for Each of 64 Clusters[a]

Maximum PE's per NE	Configuration	Total PE's	Cluster failure probability	Sufficiently low failure probability
2	$4 \times 2 + 1 \times 1$	9	1.5978×10^{-8}	No
2	**5×2**	**10**	**3.0796×10^{-12}**	**Yes**
3	$3 \times 3 + 1 \times 1$	10	1.1930×10^{-8}	No
3	**$3 \times 3 + 1 \times 2$**	**11**	**1.8716×10^{-12}**	**Yes**
4	$2 \times 4 + 1 \times 3$	11	8.0494×10^{-9}	No
4	**3×4**	**12**	**1.8008×10^{-12}**	**Yes**
5	$2 \times 5 + 1 \times 2$	12	8.5648×10^{-9}	No
5	**$2 \times 5 + 1 \times 3$**	**13**	**7.3521×10^{-11}**	**Yes**
6	$2 \times 6 + 1 \times 1$	13	9.0821×10^{-9}	No
6	**$3 \times 6 + 1 \times 2$**	**14**	**8.2332×10^{-11}**	**Yes**
7	2×7	14	9.0415×10^{-9}	No
7	**$2 \times 7 + 1 \times 1$**	**15**	**9.1622×10^{-11}**	**Yes**
8	$1 \times 8 + 1 \times 7$	15	4.8237×10^{-9}	No
8	**2×8**	**16**	**9.1557×10^{-11}**	**Yes**

[a]The probability of failure over the short-term mission for each of the 64 clusters must be

$$10^{-7}/64 = 1.5625 \times 10^{-9}$$

for intercluster communication over the ½-hour engagement mission. In this file, the failure probability of each intercluster link is modeled as the failure probability of the link, the two PE's connected to it, and their two NE's. The reader is cautioned that some failure behaviors have been modeled as independent that are definitely not independent in the interest of obtaining a quick and very rough estimate. For example, a PE failure is counted twice—in computation reliability and in communication reliability. This amounts to approximating that

$$P(A) \text{ or } P(B) = P(A) \text{ and } P(B)$$

For the small failure rates that are used in these computations, the difference between these two values is very small, at least within the same order of magnitude. However, a more accurate analysis modeling the actual failure behavior of PE's connected to NE's should be completed once a specific interconnection structure is chosen and specifically analyzed. However, since information on the behavior of the SDI battle management algorithm is still inadequate for choosing an appropriate interconnection structure, trying to model the system behavior more accurately at this stage would be pointless.

Consider a system in which each of the clusters is connected to four other clusters. From each cluster, there are at most 4 different clusters that are 1 hop away, at most $4 \times 3 = 12$ clusters that are 2 hops away, and at most $4 \times 3 \times 3 = 36$ clusters that are 3 hops away. For a system of 64 clusters, the farthest distance between 2 clusters will be at least 4 hops. The actual farthest distance between any two clusters in a configuration is dependent on the topology.

Since there are 64 clusters and each cluster is connected to 4 other clusters, there is a total of 256 intercluster communication paths. Since the system has been divided into so many clusters with only four parallel computations per cluster, a high rate and volume of intercluster communication must be assumed. Thus, it will be assumed that failure of any one of these intercommunication paths may degrade performance below acceptable levels because longer communication paths will be necessitated and message traffic will be increased on other paths. Following this assumption, the probability of failure of any one of these intercommunication paths over the ½-hour engagement mission must be less than

$$1 \times 10^{-7}/256 = 3.9 \times 10^{-10}$$

The Markov analysis shows that the use of triplex links between clusters provides a probability of failure of 3.02×10^{-8}, which is 2 orders of magnitude too high. Therefore, quad links will be required that give a probability of failure for each communication path of 2.44×10^{-12}. Using the ASSIST file shown in appendix D, it was determined that there must be 75 links in the initial configuration for each of the 256 communication paths to ensure that the probability of not having quad links after 5 years is

$$0.0001/256 = 3.9 \times 10^{-7}$$

Even though this analysis optimistically assumes that all but four of the links would be attached to cold spare PE's, this calls for a total of $75 \times 256 = 19\,200$ communication links in the initial configuration!

The preceding paragraphs have shown that the development of a feasible architecture for a highly parallel processor for a 5-year mission with no repair is probably impossible using current technology and massive processor-level redundancy as used in the FTPP. Various design trade-offs were used to try to overcome the problems. If the system is divided into only a few clusters, then the clusters are too large to support the synchronization and fast intracluster communication needed for performance. Dividing the system into more clusters leads to high intercluster communication and high numbers of spare components needed for each cluster. The problem is that since the mean time to failure of a processor is actually less than the mission time, the system must provide enough sparing capabability to tolerate the loss of most of the processors in the system. The next section shows that even the most optimistic reliability analysis of a system with processor-level redundancy cannot eliminate this problem.

A More Optimistic Long-Mission Analysis

A report produced by SoHar Incorporated for the U.S. Air Force (see Hecht and Hecht 1985) suggests that a more optimistic long-mission analysis may be possible. Analysis of over 3000 reports of anomalous incidents affecting spacecraft revealed that the failure rate of components on spacecraft may actually decrease over a long mission time. This report states that approximately 45 percent of the failures on spacecraft are due to design insufficiencies or encountering an unexpected environment. These types of failures are naturally higher near the beginning of the mission when the rate of encountering previously untested environments and operating modes is higher. According to this report, these failures do not appear to be exponentially distributed, and a Weibull distribution or some other two-parameter distribution should be used to model them. The other 55 percent of failures (consisting of the classical parts, quality, operational, and other types of

failures) were found not to deviate significantly from the exponential distribution over long mission times. The SoHar report thus recommends modeling these two types of failures independently. If insufficient information is available for accurately modeling the nonexponential failures, the report recommends simply reducing the failure rate used over long mission times by one-half.

The SoHar report (Hecht and Hecht 1985) represents a significant and optimistic departure from beliefs commonly held by reliability engineers. The results of the report have yet to be corroborated by other studies. The failure data are plotted as a function of failures per spacecraft, not failures per component. Over the lifetime of a spacecraft, a number of components fail and are taken out of the active system configuration. Thus, fewer components are available to fail as the mission continues. Also, as stated in the SoHar report, they felt that the apparent decrease in the failure rate over the long mission time of a spacecraft might be attributable to a loss of interest and diligence in reporting errors. Clearly, much more analysis of component failure rates over long mission times is needed. However, for completeness, the long-mission reliability analysis of our application is repeated below in which SoHar's more optimistic failure rates are applied.

During the 5-year dormant mission of the SDI satellite, the system will be repeating diagnostics and performing some monitoring functions, but it will probably not be encountering many new environments or operating modes. After launch and a short initial operating period, the system should be experiencing few design and environment failures. Instead of trying to model design and environment failures with a Weibull distribution or some other two-parameter distribution since we have no idea what the parameters should be, the number of design and environment failures during the 5-year mission will be assumed to be negligible. We can make this assumption because the system is essentially dormant over the 5-year mission and our major concern is availability of the system at the end of 5 years. To model this, the long-mission analyses of the previous section were repeated with the failure rates of active components multiplied by a factor of ½. The failure rate of cold spare processors is not affected by design or environment failures, and so the spare-processor failure rate was kept at the same value used previously. It should be noted that this analysis does not take into account the high number of components expected to fail during launch of the satellite into space.

For a system divided into 16 clusters, there must be 38 PE's still working per cluster after 5 years with a probability of

$$0.0001/16 = 6.25 \times 10^{-6}$$

Using the lower active failure rate requires an initial cluster configuration of 21 NE's with 6 PE's each (as compared with the previous estimate of 31 NE's). However, even a 21-NE cluster is probably too big to be feasible.

If the system is divided into 32 clusters, there must be 22 PE's still working after 5 years with a probability of

$$0.0001/32 = 3.125 \times 10^{-6}$$

With the lower active failure rate, the initial cluster configuration needed consists of 14 NE's with 6 PE's each (compared with the previous estimate of 21 NE's). A cluster of 14 NE's would require $14 \times 13/2 = 91$ intracluster links and may be feasible, depending on interconnection and physical-proximity restrictions. The total initial configuration for 32 clusters would consist of $14 \times 32 = 448$ NE's, $14 \times 6 \times 32 = 2688$ PE's, and $91 \times 32 = 2912$ intracluster links.

With 64 clusters, there must be 14 PE's still working after 5 years with a probability of

$$0.0001/64 = 1.5625 \times 10^{-6}$$

The new estimate for the initial cluster configuration is 10 NE's with 6 PE's each (compared with the previous estimate of 15 NE's). A cluster of 10 NE's and $10 \times 9/2 = 45$ intracluster links is probably feasible. The total system of 64 clusters would then consist of $10 \times 64 = 640$ NE's, $10 \times 6 \times 64 = 3840$ PE's, and $45 \times 64 = 2880$ intracluster links.

The analysis of intercluster link failures over the long mission was also repeated with a lower failure rate for active link failures by a factor of 0.5. Environments and operation modes should not significantly affect link failures, and so reducing link failure rates by one-half seems unjustified. However, we are attempting to apply the SoHar technique consistently and optimistically.

If the system is divided into 64 clusters with each cluster connected to 4 other clusters, the probability of degrading below quad intercluster links must be less than

$$0.0001/256 = 3.9 \times 10^{-7}$$

over 5 years. With the lower failure rate, the initial configuration still requires 52 links for each intercluster communication path (as compared with the previous estimate of 75 links). Thus, even with the

lower failure rate, the entire system of 64 clusters would require $52 \times 256 = 13\,312$ intercluster links.

If a cluster of 14 NE's is feasible, the system could be divided into only 32 clusters. The probability of degrading one of the $32 \times 4 = 128$ intercluster paths below a quad must be

$$0.0001/128 = 7.8 \times 10^{-7}$$

over 5 years. This would require 50 links per intercluster path for a system total of $50 \times 128 = 6400$ links.

The above analysis was extremely optimistic. All components, including communication links, were assumed to be twice as reliable as experience has shown them to be, and the high number of components expected to fail during launch of the satellite into space was ignored. Yet, even when using unrealistically optimistic failure rates, an initial configuration of 640 NE's, 3840 PE's, 2880 intracluster links, and 13 312 intercluster links is still needed.

Alternative Approaches to Reliability Over a Long Mission

As shown by the above analyses, a reasonable configuration can be designed to handle the performance and reliability needed over the ½-hour engagement mission. However, the 5-year mission time presents a big problem. Since the average life of a processor is estimated to be 20 000 hours, we must expect to lose many processors over a mission of 43 500 hours. In fact, to support 256 parallel computations over a 5-year mission requires thousands of processors. Configuring a system to tolerate the failures expected over a 10-year mission using 20 000-hour processors would be outrageous. When massive processor-level redundancy is used to tolerate huge numbers of expected processor failures, exorbitant numbers of communication links are required to interconnect the structure.

Two possible approaches to solving this problem are presented in this section. The first approach is to build more reliable processors. The second approach is to allow reconfiguration at a lower level than the processor-level reconfiguration available on the FTPP. A third possible approach, which is not pursued in this paper because it probably is not feasible for the target application, would be provision for the repair of failed processors.

More Reliable Processors

Consider a system built of PE's and NE's that are five times as reliable as those available today. With active PE failure rates of 1×10^{-5}, NE failure rates of 5×10^{-6}, and cold spare PE failure rates of 1×10^{-6}, an initial configuration of only 51 PE's and 9 NE's per cluster is sufficient for a system of 32 clusters. Total system size is then only $51 \times 32 = 1632$ PE's and $9 \times 32 = 288$ NE's. This is a much more reasonable initial configuration.

If the system could be built with PE's and NE's that are 10 times more reliable than those available today, the initial configuration could be even smaller. With active PE failure rates of 5×10^{-6}, NE failure rates of 1×10^{-6}, and cold spare PE failure rates of 5×10^{-7}, an initial configuration of only 36 PE's and 6 NE's per cluster is sufficient for a system of 32 clusters. Total system size is then only $36 \times 32 = 1152$ PE's and $6 \times 32 = 192$ NE's. These clusters are small enough that a system divided into only 16 clusters is feasible. An initial configuration of 61 PE's and 7 NE's per cluster is sufficient for a system of 16 clusters, thus giving a total system size of $61 \times 16 = 976$ PE's and $11 \times 16 = 176$ NE's. This is approximately one-sixth the size of the original configuration.

The above analyses show the importance of having more reliable processors for long missions with no repair. Unfortunately, such reliable processors are far beyond the current state of the art.

Chip-Level Redundancy

A second promising approach is to allow reconfiguration of faulty components at a lower level than the processor level. Figure 4 shows the effects of subcomponent-level redundancy on the initial configurations needed for a system of 16 clusters. In the figure, each processor has been split into one to five independent subcomponents. Each subcomponent has an equal share of the original failure rate of the component; for example, for the processor split into five subcomponents, each subcomponent has one-fifth the failure rate of the original processor.

As shown in figure 4, the original processor-level reconfiguration required an initial configuration of 126 PE's per cluster. Splitting each processor into 2 equal subcomponents reduces the initial configuration requirement to 108 PE's. Splitting each processor into 5 equal subcomponents reduces the initial configuration to only 66 PE's per cluster. Those initial cluster configurations include the 38 PE's needed to support the ½-hour mission. Thus, the number of spares needed for availability after the 5-year dormancy is decreased from 88 to only 28 PE's, a decrease of 68 percent, when each PE is separable into 5 subcomponents.

A number of factors were not taken into account in this quick analysis. First, it is somewhat optimistic to assume that a processor can actually be

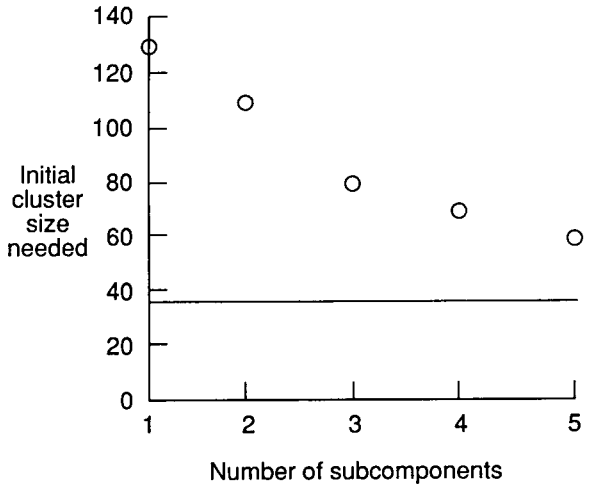

Figure 4. Number of PE's needed in initial clusters of 16-cluster system for PE's separated into 1 to 5 subcomponents. The line drawn at 38 PE's shows the minimum cluster needed for processing.

broken into a number of relatively equal pieces without any subsequent increase in the failure rate of the total component. Second, it was assumed that over the 5-year dormant mission the PE subcomponents could be fully reconfigured within a cluster to make up the maximum number of working PE's. Whether this full reconfigurability could be supported without compromising the fault-containment regions is doubtful. Third, the increases in failure rates due to the extra interconnections needed for this flexible reconfigurability are not accounted for. However, even when accounting for extreme optimism in estimation, the effectiveness of chip-level reconfiguration is clear.

Concluding Remarks

A set of methods, issues, and techniques was presented for evaluating the fault tolerance potential of parallel-processor architectures and for designing a system configuration of a fault-tolerant parallel-processor architecture to meet the reliability and performance requirements of a Strategic Defense Initiative (SDI) satellite application. Several gaps in the tools needed for an analysis of these types of architectures were identified, including tools for performance and communication network analysis.

The fault tolerance potentials of a number of proposed fault-tolerant parallel-processor architectures were briefly evaluated. The Fault Tolerant Parallel Processor (FTPP) under development at the Charles Stark Draper Laboratory, Inc., was chosen as the most promising candidate architecture. The fault tolerance aspects of the FTPP were examined, and a preliminary configuration of the FTPP to meet the requirements of the SDI defense satellite application was developed. However, it may be infeasible to launch a satellite with the large number of spare processors needed in the initial FTPP configuration to tolerate the component failures to be expected during a 5-year mission.

The design of a configuration to handle the performance and reliability needed over the ½-hour engagement mission is probably within reach with today's technology. However, without components that are significantly more reliable than those available today, a highly parallel processor architecture for a 5-year mission appears to be infeasible.

Massive processor-level redundancy is not a reasonable solution to the problem. Two promising alternative approaches were shown to have significant promise—the development of considerably more reliable components, and the use of redundancy at the chip level rather than at the processor level as used in the FTPP.

NASA Langley Research Center
Hampton, VA 23665-5225
June 7, 1989

Appendix A

Analysis of PE Failures Per Cluster During Engagement

The ASSIST input file used to analyze the failure behavior of processors and communication links within a cluster for a given cluster configuration is given below. The parameters are set to analyze a cluster responsible for 16 dual-redundant computations.

```
INPUT NE_SIZE;      (* Highest number of PEs per NE in configuration *)
INPUT N1, N2, N3, N4, N5, N6, N7, N8;
   (* Nx = Number of NEs with x PEs attached, for x = 1,2, ..., 8 *)
TIME = 0.5;          (* Half hour mission time *)
LAMP = 5E-5 + 1E-6;  (* PE permanent failure rate including its link *)
LAMT = 5E-4 + 1E-5;  (* PE transient failure rate including its link *)
LAMN = 1E-5 + 1E-6*(N1+N2+N3+N4+N5+N6+N7+N8-1);
     (* NE failure rate including its links to other NEs in cluster  *)
SPACE = (NUM: ARRAY[0..8], (* Number of NEs with how many nonfailed PEs *)
         NUMCOMPS,          (* Total number of PEs *)
         FAILED,            (* Flag indicating system failure due to coverage *)
         NCF);              (* Total number of component failures *)
START = (0, N1, N2, N3, N4, N5, N6, N7, N8,                    (* NUM array *)
         N1 + N2*2 + N3*3 + N4*4 + N5*5 + N6*6 + N7*7 + N8*8, (* NUMCOMPS *)
         0, 0);                                                (* FAILED, NCF *)
DEATHIF NUMCOMPS < 16;     (* System failure due to exhaustion *)
DEATHIF FAILED = 1;        (* System failure due to coverage *)
PRUNEIF NCF > 3;           (* Prunes model at third component failure level *)

FOR I = 1, NE_SIZE;
   IF NUMCOMPS >= 32 THEN;   (* All computations duplex *)
           (* NE failures *)
      IF NUM[I] > 0 TRANTO
         NCF=NCF+1, NUMCOMPS=NUMCOMPS-I, NUM[I]=NUM[I]-1, NUM[0]=NUM[0]+1
         BY NUM[I]*LAMN;
           (* PE failures *)
      IF NUM[I] > 0 TRANTO
         NCF=NCF+1, NUMCOMPS=NUMCOMPS-1, NUM[I]=NUM[I]-1, NUM[I-1]=NUM[I-1]+1
         BY I*NUM[I]*LAMP;
   ELSE;                     (* No spares, some computations simplex *)
           (* NE failures - uncovered simplex *)
      IF NUM[I] > 0 TRANTO FAILED=1 BY (32-NUMCOMPS)*NUM[I]*0.5*LAMN;
           (* NE failures - covered simplex *)
      IF NUM[I] > 0 TRANTO
         NCF=NCF+1, NUMCOMPS=NUMCOMPS-I, NUM[I]=NUM[I]-1, NUM[0]=NUM[0]+1
         BY (32-NUMCOMPS)*NUM[I]*0.5*LAMN;
           (* NE failures - duplex *)
      IF NUM[I] > 0 TRANTO
         NCF=NCF+1, NUMCOMPS=NUMCOMPS-I, NUM[I]=NUM[I]-1, NUM[0]=NUM[0]+1
         BY (NUMCOMPS-16)*NUM[I]*LAMN;
           (* PE failures - uncovered simplex *
      IF NUM[I] > 0 TRANTO FAILED=1 BY (32-NUMCOMPS)*I*NUM[I]*0.5*(LAMP+LAMT);
           (* PE failures - covered simplex *)
      IF NUM[I] > 0 TRANTO
         NCF=NCF+1, NUMCOMPS=NUMCOMPS-1, NUM[I]=NUM[I]-1, NUM[I-1]=NUM[I-1]+1
         BY (32-NUMCOMPS)*I*NUM[I]*0.5*LAMP;
           (* PE failures - duplex *)
```

```
      IF NUM[I] > 0 TRANTO
         NCF=NCF+1, NUMCOMPS=NUMCOMPS-1, NUM[I]=NUM[I]-1, NUM[I-1]=NUM[I-1]+1
         BY (NUMCOMPS-16)*I*NUM[I]*LAMP;
   ENDIF;
ENDFOR;
```

This ASSIST file can be used to evaluate any cluster configuration with up to 8 PE's per NE. The configuration to be evaluated is specified using the constants N1 through N8 to denote the number of NE's in the cluster having 1 through 8 PE's, respectively. The array state space variable NUM tracks the current configuration of unfailed NE's and PE's in each state of the semi-Markov model. For example, if NUM(5) has the value 3 for a state, then in that state the system has three unfailed NE's that are attached to five unfailed PE's. The state space variable FAILED is set when an uncovered failure results in system failure. The constant LAMP is the permanent failure rate for a PE plus the link attaching that PE to its NE. The constant LAMT is the transient failure rate of the PE and its link to the NE. If a computation is done in duplex, then transient failures are completely covered. However, if a computation is done only in simplex, then a transient failure of the PE or its link can cause a system failure with a probability of 0.5. The constant LAMN is the permanent failure rate of one NE of the cluster plus all the links connecting that NE to the other NE's in the cluster. Pruning at the third component failure level was used to reduce the size of the semi-Markov models generated. The largest of the models contained only 41 states and 332 transitions. The ASSIST program generated this model in only 30 sec of Central Processing Unit (CPU) time, and the SURE program solved the model in 3 CPU sec.

Appendix B

Analysis of PE Failures Per Cluster Over a 5-Year Mission

The simple ASSIST file given below models only PE failures and was used to get a rough estimate of the number of spares needed over a 5-year mission.

```
PROCS = 14;
INPUT NE;
TIME = 43800;          (* Five year mission time *)
LAMP = 5.1E-5;  (* PE failure rate (includes its link) *)
LAMS = 5.1E-6;  (* Spare failure rate *)
SPACE = (NUM_PE);
START = (NE*6);
DEATHIF NUM_PE < PROCS;
IF NUM_PE > 0 TRANTO NUM_PE=NUM_PE-1 BY (PROCS*LAMP)+((NUM_PE-PROCS)*LAMS);
```

Appendix C

Analysis of Intercluster Communication During Engagement

The ASSIST file given below was used to get a rough estimate of the number of links needed for intercluster communication over the ½-hour engagement mission. This model determines the probability of failure of the set of redundant intercluster links between two clusters. The failure probability of each intercluster link is modeled as the failure probability of the link, the two PE's connected to it, and their two NE's. Since the probability of failure of the intercluster network determined by this model includes some PE failures, adding the failure probability from this model to those obtained from the model described in appendix A would give a very conservative system-failure probability because PE failures would be counted multiple times. However, clearly the probability of system failure due only to intercluster failures must be less than the total system-failure probability, and so this model is useful for roughly estimating the number of redundant intercluster links needed.

It would have been considerably more accurate to choose a specific interconnection structure and model the entire architecture with the actual failure behavior of PE's connected to NE's. However, this would have required many iterations to find an optimum interconnection strategy and would have resulted in very large Markov models that take hours to solve. It is much more practical to use very simple models during the first preliminary analysis.

```
INPUT NL;
TIME = 0.5;  (* Half hour mission time *)
LAMP = 12.1E-5;  (* Failure rate of link includes its 2 PEs & their NEs *)
LAMT = 10*LAMP;

SPACE = (NUML,FAILED);
START = (NL,0);
DEATHIF FAILED = 1;
IF NUML > 2 TRANTO NUML = NUML-1 BY NUML*LAMP;
IF NUML = 2 TRANTO NUML = NUML-1 BY 0.5*(LAMP+LAMT);  (* C = 0.5 *)
IF NUML = 2 TRANTO FAILED = 1 BY 0.5*(LAMP+LAMT);
IF NUML = 1 TRANTO FAILED = 1 BY LAMP+LAMT;
```

Appendix D

Analysis of Intercluster Communication Over a 5-Year Mission

The ASSIST input file used to describe the failure behavior of intercluster links over a 5-year mission is given below. This file models the probability of degradation of the set of intercluster links between two clusters to below the quad redundancy needed for the ½-hour engagement mission. This simple model makes a very conservative assumption that all the redundant links except for the four active ones are attached to cold spare PE's on cold spare NE's, and thus they have lower failure rates.

```
INPUT NL;
TIME = 43800;       (* Five year mission time *)
LAMP = 12.1E-5;     (* Each includes the 2 PEs, their NEs plus one link *)
LAMS = 12.1E-6;     (* Failure rate for spare *)
SPACE = (NUM_L);
START = (NL);
DEATHIF NUM_L < 4;
IF NUM_L > 0 TRANTO NUM_L=NUM_L-1 BY 4*LAMP + (NUM_L-4)*LAMS;
```

54

References

Baker, Robert; and Scheper, Charlotte 1986: *Evaluation of Reliability Modeling Tools for Advanced Fault Tolerant Systems.* NASA CR-178067.

Bavuso, S. J.; and Peterson, P. L. 1985: *CARE III Model Overview and User's Guide (First Revision).* NASA TM-86404.

Butler, Ricky W. 1988: *A Survey of Provably Correct Fault-Tolerant Clock Synchronization Techniques.* NASA TM-100553.

Butler, Ricky W.; and Stevenson, Philip H. 1988: *The PAWS and STEM Reliability Analysis Programs.* NASA TM-100572.

Butler, Ricky W.; and White, Allan L. 1988: *SURE Reliability Analysis—Program and Mathematics.* NASA TP-2764.

Cohen, G. C.; Lee, C. W.; Brock, L. D.; and Allen, J. G. 1986: Design Validation Concept for an Integrated Airframe Propulsion Control System Architecture (IAPSA II). NASA CR-178084.

Dugan, Joanne Bechta; Trivedi, Kishor S.; Smotherman, Mark K.; and Geist, Robert M. 1986: The Hybrid Automated Reliability Predictor. *AIAA J. Guid.,* Control & Dyn., vol. 9, no. 3, May–June.

Dzwonczyk, M. J.; McKinney, M. F.; Adams, S. J.; and Gauthier, R. J. 1989: *Avionics Architecture Studies for the Entry Research Vehicle.* NASA CR-181828.

Friend, Steven Alan 1986: *Process Synchronization Within a Loosely Coupled Fault Tolerant Parallel Processing System.* CSDL-T-937, Charles Stark Draper Lab., Inc., Dec.

Harper, Richard Edwin 1987: *Critical Issues in Ultra-Reliable Parallel Processing.* CSDL-T-944, Charles Stark Draper Lab., Inc., June.

Hecht, Herbert; and Hecht, Myron 1985: *Reliability Prediction for Spacecraft.* RADC-TR-85-229, U.S. Air Force, Dec. (Available from DTIC as AD A164 747.)

Jagannathan, R.; and Ashcroft, E. A. c.1986: Fault-Tolerant Aspects of the Eduction Model and Architecture. *Proceedings—IEEE/AIAA 7th Digital Avionics Systems Conference,* IEEE Catalog No. 86CH2359-8, pp. 515–522.

Johnson, Sally C. 1986: *ASSIST User's Manual.* NASA TM-87735.

Johnson, Sally C. c.1988: Reliability Analysis of Large, Complex Systems Using Assist. *A Collection of Technical Papers—AIAA/IEEE 8th Digital Avionics Systems Conference,* pp. 227–234. (Available as AIAA-88-3898-CP.)

Kern, George A. c.1978: Operational Influences on Avionics Reliability. *1978 Proceedings Annual Reliability and Maintainability Symposium,* IEEE Catalog No. 78CH1308-6R, Inst. of Electrical and Electronics Engineers, Inc., pp. 231-242.

Krishna, C. M.; Shin, Kang G.; and Butler, Ricky W. 1985: Ensuring Fault Tolerance of Phase-Locked Clocks. *IEEE Trans. on Comput.,* vol. C-34, no. 8, Aug., pp. 752–756.

Lala, Jaynarayan H. c.1984: Advanced Information Processing System. *Proceedings of the AIAA/IEEE 6th Digital Avionics Systems Conference,* pp. 199–210. (Available as AIAA-84-2644.)

Lala, J. H.; Alger, L. S.; Gauthier, R. J.; and Dzwonczyk, M. J. 1986: *A Fault Tolerant Processor To Meet Rigorous Failure Requirements.* CSDL-P-2705, Charles Stark Draper Lab., Inc., July.

McGough, John G.; and Swern, Fred L. 1981: *Measurement of Fault Latency in a Digital Avionic Mini Processor.* NASA CR-3462.

Military Handbook 1982—Reliability Prediction of Electronic Equipment. MIL-HDBK-217D, U.S. Dep. of Defense, Jan. 15. (Supersedes MIL-HDBK-217C, Apr. 9, 1979.)

Pease, M.; Shostak, R.; and Lamport, L. 1980: Reaching Agreement in the Presence of Faults. *J. ACM,* vol. 27, no. 2, Apr., pp. 228–234.

Rasmussen, R. D.; Bolotin, G. S.; Dimopoulos, N. J.; Lewis, B. F.; and Manning, R. M. c.1987: Advanced General Purpose Multicomputer for Space Applications. *Proceedings of the 1987 International Conference on Parallel Processing,* Sartaj K. Sahni, ed., Pennsylvania State Univ. Press, pp. 54–57.

Rennels, David A. c.1986: On Implementing Fault-Tolerance in Binary Hypercubes. *16th Annual International Symposium on Fault-Tolerant Computing Systems—Digest of Papers,* IEEE Catalog No. 86CH2335-8, IEEE Computer Soc., pp. 344–349.

Schabowsky, Richard S., Jr.; Gai, Eliezer; Walker, Bruce K.; Lala, Jaynarayan H.; and Motyka, Paul c.1984: Evaluation Methodologies for an Advanced Information Processing System. *Proceedings of the AIAA/IEEE 6th Digital Avionics Systems Conference,* pp. 217–224. (Available as AIAA-84-2647.)

Troxel, Gregory D. 1987: *Detection of and Recovery From Deadlock in a System Using Remote Procedure Calls.* CSDL-T-959, Charles Stark Draper Lab., Inc., June.

Wilson, Ron 1987: Designers Rescue Superminicomputers From I/O Bottleneck. *Comput. Design,* vol. 26, no. 18, Oct. 1, pp. 61–71.

NASA
National Aeronautics and Space Administration

Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TM-4123		

4. Title and Subtitle	5. Report Date
Evaluation of Fault-Tolerant Parallel-Processor Architectures Over Long Space Missions	August 1989
	6. Performing Organization Code
7. Author(s)	8. Performing Organization Report No.
Sally C. Johnson	L-16437
9. Performing Organization Name and Address	10. Work Unit No.
NASA Langley Research Center Hampton, VA 23665-5225	505-66-21-01
	11. Contract or Grant No.
12. Sponsoring Agency Name and Address	13. Type of Report and Period Covered
National Aeronautics and Space Administration Washington, DC 20546-0001	Technical Memorandum
	14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

This paper examines the impact of a 5-year space mission environment on fault-tolerant parallel-processor architectures. The target application is a Strategic Defense Initiative (SDI) defense satellite requiring 256 parallel processors to provide the computation throughput. The reliability requirements are that the system still be operational after 5 years with a probability of 0.9999 and that the probability of system failure during ½ hour of full operation be less than 10^{-7}. The fault tolerance features that an architecture must possess to meet these reliability requirements are presented, many potential architectures are briefly evaluated, and one candidate architecture, the Fault Tolerant Parallel Processor (FTPP) under development at the Charles Stark Draper Laboratory, Inc., is evaluated in detail. A methodology for designing a preliminary system configuration to meet the reliability and performance requirements of the mission is then presented and demonstrated by designing an FTPP configuration.

17. Key Words (Suggested by Authors(s))	18. Distribution Statement
Reliability analysis Fault tolerance Parallel processors	Unclassified–Unlimited Subject Category 65

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	31	A03

For sale by the National Technical Information Service, Springfield, Virginia 22161-2171