PARALLEL ARCHITECTURES FOR PLANETARY EXPLORATION REQUIREMENTS

(P.A.P.E.R)

(Interim Report)

Principal Investigators

Ruknet Cezzar and Ranjan K. Sen
Department of Computer Science
Hampton University
Hampton, Virginia

Technical Officer

Wayne H. Bryant, Head
Systems Architectures Branch
NASA Langley Research Center
ISD M/S 478
Hampton, Virginia 23665

Grant Number: NAG-1-949

Grant Period: January 12, 1989 - December 31, 1989

June 25, 1989

TABLE OF CONTENTS

I. OVERVIEW

The Department of Computer Science at Hampton University, historically a minority institution, was established in 1985. This fairly new department offers B.S. degree in Computer Science and Computer Information System. The department is accredited by the National Board and both the programs mentioned above closely adheres to the Association for Computing Machinery (ACM) guidelines.

In the department a considerable research effort is being made through projects funded by organizations like NASA Langley Research Center, Department of Defense (DoD), and the National Science Foundation (NSF). Some of these projects, such as the planning project with NSF (coined HELPAR), aims to develop a long-term research infrastructure. The existing research emphasis emanates from the interested faculty and also from the desire for establishment of a graduate program in computer science, which is presently absent.

Parallel Architectures for Planetary Exploration Requirements (PAPER) project is essentially research oriented towards technology insertion issues for NASA's unmanned planetary probes. This has been initiated to complement and augment the long-term efforts for space exploration with particular reference to NASA/LaRC's (NASA Langley Research Center) research needs for planetary exploration missions of mid and late 1990s. The rationale in this project includes the prospect of building research expertise involving minority faculty and students in this department.

This report is organized as follows. Section II essentially covers the activities of the project personnel carried out since January 1989. These include initial efforts in setting up the administrative and related mechanisms for smooth functioning. This section also includes the discussions of our participation in technical seminars, conferences, literature surveys.

Section III presents our basic review of the requirements data for the "PAPER Class architectures". We review the old AIPS (Advanced Information Processing Systems) requirements that were the result of a survey conducted in 1983 by CSDL, and more recent requirements data involving planetary rovers from Jet Propulsion Laboratory. The discussion includes review of experimental fault-tolerant architectures proposed by Charles Stark Draper Laboratory (CSDL) and Jet Propulsion Laboratory/California Institute of Technology (JPL/Caltech). Specifically, these include CSDL's AIPS Configuration based on Fault Tolerant Processor (FTP) building blocks, JPL's MAX Fault-Tolerant Dataflow Multiprocessor, and a Large-Scale Fault-Tolerant Snooping-Cache Multiprocessor also developed at JPL/Caltech.

Section IV presents the areas of research focus for the investigators. It is felt that, while the requirements data is being analyzed, these will be major impact items for designing computational support for planetary exploration missions. Specifically, we cover the problem of reconfiguration for distributed fault tolerance, applicative caching for reccursive

computations, software fault-tolerance and artificial intelligence support for reliability. In addition, as an early preparation for Phase II, we review the promising new area of neural computing and its possible use in space explorations. A bibliography of recent publications along with a survey is provided. In addition, ways of implementing neural networks on distribute systems, such as the planetary rover [64] is discussed.

Section V outlines the plans for the remainder of Phase I. This includes the planned activities involving on-the-spot study CSDL and JPL's work in the area of computer architectures for unmanned deep space probes. This section concludes by defining our research directions for the remainder of the current phase. These research directions will lead us into the continuation of the project into subsequent phases.

Section VI concludes by stating the additional work to be done in the remainder of Phase 1, and potential research areas for subsequent phases with renewed funding. It was possible only to look into two different systems in somewhat details. Related architectures that are current projects in NASA Research Laboratories and other National Laboratories need to be examined.

II. ACTIVITIES

As already mentioned a major part of our activities in the interim period have been administrative. The purpose was the building of the faculty research team with task assignments and enlisting the participation of graduate and undergraduate minority students. For LaRC side, we have gathered information regarding the requirements for computational support, and possible parallel architectures for, the unmanned planetary exploration missions.

In this period the efforts have been focused upon AIPS work being done at Charles Stark Draper Laboratory, and the various JPL initiatives that involve novel fault tolerance concepts and specification of requirements of a planetary rover.

2.1 Getting Started

One of our first steps in planning was to provide details of our travel plans for visits to NASA sites in order to collect direct information regarding requirement and evaluation process. Upon funding for Phase 1 of the project for the period from January 12 to December 31, 1989, we had solicited the participation of graduate and undergraduate students in Hampton University. Once we obtained such support we provided, generally in writing, task assignments for both the student assistants and research faculty of the project. The details of task assignments, shown in Appendix I, present the nature of involvement of such people.

With the help of the student assistants, and through partial effort by the co-principal investigators, we have collected the relevant articles, proceedings, and textbooks. In particular, we have organized the references included in our original proposal. Moreover, as a preparation for future research on neural computing, we have collected various tutorial and investigative material for survey and identification of problem areas. This included a tutorial series of articles that appeared in AI Expert, and a bibliography of the most recent books and articles shown in Appendix II.

Another major effort in this interim period was the building of a multiple processor performance simulator for testing out various process management policies for parallel machines [23,24]. This work was carried out by two student assistants using Hampton University's Academic Computer Center VAX facilities. The simulator is being developed in VMS Pascal. So far we had a modest success because of slowing down due to change in personnel. When completed this simulator will flexibly test out process management policies for efficient distributed computations. At the same time, we have obtained and installed the iPSC/2 simulator for hypercube programming experiments on the VAX 8350 ULTRIX system [50].

Another effort was in the researching of applicative caching on list oriented dataflow machines [92]. Moreover, research in the area of graph-theoretic computation were conducted [93]. These works have been presented to seperate International conferences.

2.2 Conferences, Seminars, and Presentations

The co-investigators, with funding from an NSF grant, attended the Fourth Hypercube Concurrent Computer and Applications Conference (HCCA4), Monterey, California, March 6-8, 1989. Since this conference was sponsored, in part, by JPL/Caltech, we had a chance to meet with JPL research staff and discuss related work. For example, a poster presentation [49] was of some interest. We have also met with other JPL technical staff [84] who has participated in the conference as session organizers [66] or presenters and discussed our project.

Dr. Ranjan K. Sen, co-investigator, presented his two seperate works respectively at the 20 Southeastern Conf. Combinatorics Grpah Theory Computing, Boca Rotan, Florida, Feb 1989 and at the International Conference on Computers and Information, May 23-27, Toronto, Canada. These works [92,93] concerns graph embedding and applicative caching for avoiding recomputation in highly computational jobs. We feel that these work has potential application in problems of embedding and avoiding recomputation for dataflow systems with particular reference to the MAX architectural concept [81].

We have submitted the work entitled "An Algorithm for Optimal Communication in a Distributed System with Limited Number of Ports per Node," for conference presentation. This work has relevance in terms of increasing the communication efficiency of distributed computations in a fully-connected tightly-coupled parallel architectures.

III. PAPER CLASS ARCHITECTURES

The "PAPER Class Architecture" stands for a computing system that is most suitable for unmanned planetary and deep-space explorations. Generally, this involves the three mission modules: Spacecraft (fly-by or orbiter), lander, and rovers. An important consideration, as with all other types of missions manned or unmanned, is the reliability of the computational support. This, in fact, has made the computational support of space missions more specialized and expensive. Unfortunately, there is no standard and agreed-upon terminology in this area. In general, as is the case with the discussion of AIPS and MAX systems, the terms fault-tolerance, availability, and maintainability are used to characterize the reliability of computational support. In order to clarify terminology, we first provide the following definitions.

Reliability:

The conditional probability that a system has survived the interval (0,t) provided that it was correctly operating at time t=0. Informally, this means the system is still providing the acceptable level of service at time t, after possibly undergoing faults and recoveries.

This measure is a probability usually expressed as a percentage. For example, in [83], the reliability of one VLSI chip is assumed to be .9948 (or 99.48 %) over a 10-year mission duration.

Failure Rate:

The probability of failures per unit time, where time is usually measured in hours or seconds.

For example, the failure rate of the above VLSI chip is $.6 \times 10^{-7}$ per second. Another example is that, in AIPS one mission requirement is the failure probability of 10^{-9} per hour.

Availability:

The probability the system is operational (not necessarily providing the expected functionality) at a given instant of time, independently of previous faults.

Fault tolerance:

The ability to maintain an acceptable level of system functionality with high probability over the duration of the mission in the presence of faults.

Fault tolerance is expressed in terms of the types of faults covered in a qualitative manner (e.g. Byzantine resilience means protection from malicious faults, as opposed to other types of tolerance), and also in terms of the percentage of coverage of possible multiple faults. As can be seen, it is difficult to measure and compare one "fault tolerance" with any other "fault tolerance".

Maintainability:

The extent to which the system can continue operating through replacement of modules and parts as necessary with least amount of effort.

As can be seen this is an entirely qualitative (subjective) measure that depends on the manner of module repair and/or replacement.

Dependability:

The quality of the delivered service such that reliance can justifiably be placed on this service.

As can be seen, this is a highly circular definition from [65] based on the term "reliance".

An objective of Phase 1 of our project is the specification of a PAPER class architecture (or a family of such architectures) as the basis for more focused research in subsequent phases.

In the original proposal, we have articulated and stressed the need for a systematic top-down approach for derivation of PAPER class through the following steps:

- Mission requirements
- Functional requirements
- Architectural features
- Architectural design specifications

Although, at that time, we were less familiar with the early studies on AIPS [1,2,62], it turns out that our approach is similar to this valuable but now dated work. The CSDL follow-up work on AIPS as reported in [34,82] aims to continue and extend the same approach using more recent technology in hardware and software.

Briefly, in that work, the basic approach consists of the following steps that relate to our aforementioned approach.

- **Requirements**: This is the same as the mission requirements above except for broader scope that encompass not only the unmanned planetary explorations but many other types of space missions.

- **Attributes**: We referred to these as architectural features.

- **Architectural design rules and specification guidelines**: In our case, this is simply the specification of the PAPER architecture using different techniques.

In [34], the last step involves "AIPS Knowledge Base Methodology", where well-defined design rules (in small chunks) are derived and incorporated into a comprehensive knowledge database. Presumably, this database can then be intelligently analyzed by a rule-based expert system, to arrive at system architectures that adhere to predefined reliability requirements. Although some flexibility, in terms of different architectures for different missions, exists; the overall computing system is confined to AIPS technology. AIPS technology, in turn, is based upon the FTP concepts [61].

There have been various work in similar direction elsewhere that utilizes different technologies to achieve same level of reliability. Thus, there is no need to rely exclusively upon a specific methods such as those used in FTP to build sufficiently reliable systems. We need the kinds of methods that allows utilization of various differently types of technologies to achieve the reliability as well other system requirements.

3.1 Review of Requirements Data

One of the important source materials for requirements data in connection with space missions is the AIPs study [1]. This study includes some relevant requirements data on "Unmanned Deep Space Probes" that is of interest to us. The other most important sources is the work based at JPL/Caltech involving the requirements for Planetary Rover [64] along with the fault-tolerance requirements for Spaceborne Parallel Computers [25].

We first cover the requirements data for planetary explorations in general. Then, we review of AIP Requirements Study as it pertains to unmanned deep-space probes. Finally, we briefly review the more recent work at JPL/ Caltech on computational and data storage requirements for planetary rovers and fault tolerance requirements for space borne parallel computers.

For unmanned planetary explorations, we have the following categories of requirements as listed in our proposal:

1. Ultra-high reliability
2. Availability
3. Real-time response
4. High-performance
5. Adaptability
6. Extendibility
7. Intelligence
8. Compactness
9. Robustness
10. Communication interfaces

The early work conducted in 1983-84 on Advanced Information Processing System (AIPS) requirements by Charles Stark Draper Laboratory (CSDL), although a very worthwhile effort in this direction, does not address all the requirement categories mentioned above for the unmanned planetary and deep space probes. This study attempted to determine widely-differing and

sometimes conflicting demands of seven different types of missions, with results pertaining to unmanned deep space probes not sufficiently detailed. This summary is restated below:

Allowable mission failure probability	10^{-2}
Mission duration	5 years
Mean-time-to-repair:	
On-board	Not Applicable
Ground	Not Applicable
Throughput (MIPS)	0.5
On-line Memory (Bytes)	300K
Mass memory (Bytes)	1M (long pole 100 M)
Peak I/O Rate (bits/sec)	1.5 M
Maximum cycle rate (Hz)	20
Minimum transport lag (ms)	10

As is quite obvious, these are rather modest requirements that can easily be met by today's computing technology. Even when we consider the "long pole" mass memory requirement of 100 MB), we see that the 32-bit high-end PC's (let alone 64-bit super high end PCs as in [68], meets and even exceeds the stated requirements, with the possible exception of ultra reliability through fault-tolerance. However, the reliability requirements for mission class involving planetary and deep-space probes is not very stringent. This raises the possibility a multitude of different system configurations, hardware and software, equally capable of supporting this mission class. Thus, the current hardware and software technologies seem to provide support for a wider spectrum of mission requirements. The question then becomes one of economy, efficiency, and sophistication in carrying out the typical functions.

More stringent requirements have arisen as a result of technological advancements in the following areas:

- **Sensor data capture and analysis.**

  100s of MIPS, even GFLOPS are required for real-time capture and quick (on-board) analysis of vast amount of data to be collected.

- **Robotic systems**

  The requirements are demanding, such as 10-20 MIPS for various sophisticated robotic functions (vision and cognition, locomotion, voice analysis, etc.). These requirements are especially relevant to missions where the rovers on planetary surface are used.

The JPL/Caltech work reflects the above-mentioned trends in requirements. The following is a functional breakdown for a planetary rover in terms of storage and performance requirements.

Function	Storage (Mbits)	Mops per cycle	Cycles per meter	Mops per meter
Structured Light Vision	109	6.5	5	32.5
Stereo Imaging	337	1000	0.5	500
Laser Scanner	8.2	118	5	590
Radar Sensor	1.95	10	0.2	2
Path Planner	80	5	0.01	0.05
Geometric Hazard Detection	121	27	5	135
Vehicle/Mobility Control	215	2.2	5	11
Articulation & Wheel Control	228	0.0661	5	0.330
Pointing Control	0.008	0.006	N/A	N/A
Manipulator Control	421	2.25	N/A	N/A
Telemetry Handling	634	0.75	N/A	N/A
System Fault Protection	> 1000	34	5	170
Command and Data Handling	8	1	N/A	N/A
Power & Thermal Management	0.004	0.001	N/A	N/A
Science	54000	?	N/A	N/A

Based upon the above metrics, one can determine the Mops (Mega operations per second) for each function, and then combine these judiciously to get the overall storage and efficiency requirements for an operational scenario involving a planetary rover. An example in [83] gives the total number of operations for 30-second traverse of a typical application as 848.9 Mops. An estimate of processing rate is 28.3 Mops/second. Since it is assumed that an operation on the average can be carried out by a 32-bit machine instruction, we can have a rough idea of performance requirements in the range from 20 to 30 MIPS (e.g. 28.3 for this scenario). This agrees with the above-mentioned requirements for robotics system support for rovers.

For more demanding applications, such as a laser scanner that raster scans a scene and measures range through phase modulation, the computational requirements can be as high as 118 MFLOPS per cycle for a total of 590 Mops per meter [64]. This agrees with the above data capture requirements.

3.2 Experimental Architectures

We have looked at three experimental architectures that aims to meet the above mentioned requirements, with primary focus on reliability through fault tolerance:

- o AIPS Proof of Concept Configuration
- o MAX Fault-Tolerant Multiprocessor Configuration
- o Fault-Tolerant Snooping-Cache Multiprocessor

The last experimental architecture [83] uses a cost-effective approach to the design of a fault-tolerant parallel system for ground and satellite applications. The basic approach is to add fault tolerance capability to a snooping-cache multiprocessor that already possess other desired characteristics such as high performance. This is a cost-effective solution since the parallel machine is commercially available. Specifically, Multimax/Ultramax reliability modeling is used to achieve the desired fault tolerance in Encore's Multimax parallel computer. This work is useful in providing insight into the issues of fault tolerance in a shared-memory system that has desired performance but is not fault tolerant. However, as explicitly stated in [83], this experimental system is intended as ground-based or satellite-based system, and therefore not suitable for planetary and deep space probes. One important contribution of this work is the ball park fault-tolerance requirement of 95 % reliability over a 10 year period for all space missions. Such a reliability objective is applicable to planetary exploration and deep-space probes whose mission period range from 5 to 15 years.

The last experimental system thus eliminated, either one of the first two system is a good candidate for a PAPER class. Since the design approaches for these systems mimic the proposed systematic approach from requirements to features to design, these systems can be adopted as platforms for the future planetary explorations. The advantage will be in not repeating the steps for specification of a PAPER class architecture that meet all of the ten requirement categories.

AIPS Proof of Concept Configuration

We have already mentioned the early work done in 1983 for requirements and system definition of Advanced Information Processing System. The recent presentation of AIPS Activities [34] has stated the overall objective to be the development of a knowledge base that will allow validated fault tolerance on distributed system architectures. This is to apply towards a broad range of situations including those which has a failure probability requirement as low as 10^{-9} at 10 hours.

Toward the stated aim of Proof of Concept (POC) design that is verifiable for ultra reliability and is distributed, the status of work completion has been reported in [34] as:

Distributed Hardware

The centralized AIPS configuration is shown in Figure 1. It is a 15-node configuration that consists of five triplex FTP sites, and can be configured in various redundancy formulations. As shown in Figure 2, the basic AIPS Intercomputer Network may be configured as four processing sites that consists of two triplex, one duplex, and one simplex FTP sites. The AIPS architecture consists of the following hardware building blocks:

- Fault Tolerant Processor (FTP)
- Input/Output Network that consists of up to 30 circuit-switched nodes.
- GPC-Network Interfaces consisting of Input/Output Sequencers.

Centralized Software

Consists of the Operating System and Network Services Software. The software building blocks are:

- Local System Services
- Input/Output System Services

The work on distributed software has not yet been reported. This work is currently under way for Fiscal Year 1989.

The main characteristics of the AIPS IC Network are:

- Triple redundancy of intercomputer links (layers)
- Support of mixed redundancy of subscribing sites
- Dynamic mastership contention; arbitration distributed
- Asynchronous operation (no common clock)
- Reconfigurable switching nodes in layers
- Standard bit-oriented communication protocol

The design objectives for the FY88 tasks have been summarized in two main categories as:

Quantitative --	Function reliability Function throughput Transport delay I/O Rate
Qualitative --	Adaptability and growth Maintainability Validability

Since some of the terms above have not been adequately defined, it is difficult to assess whether all these objectives have been met. However, in terms of some of the requirement items, this work has significant contribution towards definition of the PAPER class architectures. In particular, the proposed architecture integrates the fault-tolerance requirements to the computing architecture in a robust manner. This means that it meets or even exceeds the modest reliability requirements of planetary explorations

Thus, while CSDL's Distributed AIPS Configuration meets the reliability requirements, the proposed design is complex, specialized, and expensive because of stringent fault-tolerance requirements. The implementation mechanism are elaborate and usually entails great deal of hardware redundancy and software reconfiguration overhead.

MAX Fault-Tolerant Multiprocessor Configuration

At this time, we have only sketchy information from [81] regarding this experimental system that appears to be very promising for computational support of planetary explorations. As will be discussed in Section 5.1, we intend to visit JPL/Caltech in order to obtain more direct data regarding this system as well as other experimental systems that are candidates for PAPER Class.

MAX provides a great deal of flexibility in its hierarchial architecture shown in Figure 3, and meshwork configuration shown in Figure 4. Figure 3 shows the basic fault-tolerance concept at the node module, where "A MAX Module" is a duplex configuration with backup capability. As we interpret it, a programming model that consists of a mixture of data flow graph and sequential program segments is provided for maximal software reliability. This approach shown in Figure 3 allows large grain implementations of functions, data segments, and high level language constructs as tokens of a well-defined dataflow graph thereby avoiding side effects. Except for the drawback of programming complexity that must be faced in any type of data flow model, this approach is ideal for reliability above hardware levels. In addition, a hierarchial fault coverage is provided where each level protects the one beneath it. Max has some other attractive features such as efficient sparing techniques for redundant hardware components and software modules. The architecture does not claim to be ultra reliable with

Byzantine resilience and source congruency as is the case with AIPS configuration. However, it is quite capable of acceptable levels of reliability. This is accomplished by just the required degree of redundancy with respect to both hardware and software. Moreover, through partitioned resources on modules executing critical functions, mixed coverage in shared hardware is provided.

In conclusion, if we take the view that unmanned planetary missions do not necessarily require ultra reliability, the MAX architectural concept appears to be promising. We would like to obtain details of this work during our visit to see if it can adequately meet all of the ten requirement items (e.g. including size, wiring, etc.) mentioned in the previous section. We would like to investigate the addition of neurocomputing capability to this architecture as discussed in Section 4.4. If MAX architecture has neural computing capability, it can meet (or exceed) the demanding cognition task of a planetary rover as exemplified with a scene analysis in [64].

IV. AREAS OF FOCUS

It appears that there is no easy solution to the complete specification of the PAPER class architectures. This is due to the diverse nature of the requirements and inter-dependence between requirements and rapid technological advancements. In other words, requirements keep changing as the new technologies offers functional capabilities not envisaged earlier. The examples involving early AIPS requirements work (very modest computing resources) versus the recent JPL studies (extensive data acquisition requirements for scene analysis) attest to this viewpoint.

Therefore, rather than limiting our work to specific computational models, we feel it is important to focus our attention towards some crucial <u>impact issues</u> involving design of computing systems for planetary explorations. This motivated us to look at a wide range of relevant topics that will have significant impact on the future approaches to such designs.

We present below such topics in the format of research projects that can be supervised and conducted within a reasonable time period under the current funding level. We have chosen these projects in such a way that they can be integrated within the overall objective of the project.

The key tasks to be undertaken as impact items in this project has identified by us as:

1) Reconfigurability for distributed fault-tolerance,
2) Applicative caching for efficiency,
3) Software fault-tolerance involving design diversity and N-version programming,
4) Distributed neural computations on parallel systems,

We intend to carry out the initial work on the above topics during the remainder of Phase 1. With renewed funding in subsequent phases, further works can be conducted. We strongly believe that these will greatly affect the definition of the PAPER class of architectures.

4.1 Reconfiguration for Distributed Fault Tolerance

Background

Significant research in the last two decades in the design of fault-tolerant systems for spacecraft control resulted in the development of several systems like FTMP[61], SIFT[99], and FTP[34,41,42,47]. Both the aspects of centralized and distributed computation and control have been used in these systems. In all these fault-tolerant systems once a fault has been detected and diagnosed, the step of reconfiguration is of utmost importance. This becomes all the more important in space mission applications where the aspects of physical dimensions, weight, response time and fault tolerance becomes pertinent.

An unified approach for hardware and software fault tolerance[61], has been taken up in building FTP fault-tolerant system at the Draper Laboratory in Cambridge, Massachusetts. Ongoing research and development is focusing attention to building a distribute version of that system. An earlier work on designing an ultra-reliable fault-tolerant system used a design approach termed Software Implemented Fault Tolerance(SIFT) [99]. It had the objective of constructing a flexible system that can easily adapt to changes in problem specification. Fault detection and diagnosis and subsequent system reconfiguration are performed by software. Loose synchronization, in contrast to tight in FTP, is used for independent execution of tasks by separate processing units. The reliability of SIFT was proved by mathematical methods.

The software based fault-tolerance provides a more flexible system. A task is replicated and run on a varying number of processors, scheduled in a more loose manner, depending upon its degree of criticality. A global executive reconfigure the system in case a fault is diagnosed by appropriately changing the task allocation.

At the Jet Propulsion Laboratory, Pasadena, California, efforts in designing distributed fault-tolerant system started in 1973. Recently, interesting results have been obtained in distributed space-craft sequencing application using the JPL/Caltech Mark III hypercube concurrent computer. Speedup of 1.9 with two and of 3.1 with four processors have been recently reported in spacecraft sequencing application for the Mark III hypercube system[55]. The hypercube concurrent system is highly attractive because of its regular structure and upgradability. Considerable research has already taken place in designing fault-tolerant hypercube concurrent system for highly critical fault-tolerant applications[84,85,86]. As can be seen in all these works the question of efficient reconfiguration strategy in highly important.

An earlier research work on implementing fault-tolerance on hypercube concurrent systems the need for self-checking design of high-performance processors, partitioning to provide redundant sparing, and hierarchic design approaches to achieve long unmaintained life that is typical of unmanned space-crafts were observed.

The inherent redundancy of the hypercube allows the system to degrade and operate with failed processors by redistributing the computing load and redirecting information around failed nodes. It is attractive because it requires no additional hardware modifications, it is application-specific and is effective for some applications (in particular, cases which are not constrained by timing of intercommunications and where a periodic maintenance is available).

According to [85] the approach to allocate processors as spares within the existing network has some serious limitation in that recovery from permanent faults changes the topology of the machine. In the case of long-life unmaintained applications many nodes and links will fail thus making this approach impractical. A solution that is based on increasing the dimension of the hypercube by augmenting it with an additional port to make it possible to have fault-tolerance was suggested. Each of the several spare processors has a set of associated VLSI crossbar circuits attached.

A hypercube with 2^N processors can be viewed as a set of 2^s sub-cubes, each with 2^m processors (where s+m=N). It is possible to provide one spare for each subcube reachable, as before, through the N+1th dimension. If any processor fails in the subcube, the spare will have to be connected to m neighboring processors in that subcube, and one processor in each of s external subcubes to which it is connected.

The overall scheme provides simple reconfiguration algorithm and low communication delays when a failure occurs. They are suitable for applications where periodic maintenance is possible but are not recommended for long life unmaintained systems.

In long life unmaintained systems one must have methods of fine partitioning so that no single module has a high failure rate, and a typical redundant spare part is capable of backing up a large number of working parts. Typical system requirements are a 95% or greater probability of surviving five years or more. According to current technology a space computer with several megabytes of memory would have a five year reliability of less than a few percent[73].

In pursuance of constructing such systems it was proposed that the inherent multi-level redundancy of hypercube be utilized. Also better sharing of bus line, code and data will improve system reliability while maintaining hypercube structure logically.
An approach of spare allocation and reconfiguration using parameterized data distribution on fault-free subcubes and shifting workload on a faulty processor to other fault-free ones have been reported[88]. An uniform partitioning of tasks is attempted where several tasks scheduled for several virtual processors are actually distributed almost evenly onto physical

processor whose number is less that of the virtual processors. When a fault disables a processor all virtual processors are shifted to the neighboring processors evenly. This increases the communication cost (but minimally).

Recently, two schemes for reconfiguration in a hypercube concurrent computer in the presence of faults have been reported[14,15,16]. The first scheme is a hardware modification where a set of spare processors are distributed in the hypercube and the problem reduces to finding a way of allocating the spares efficiently. The second scheme of reconfiguration uses no spare processors and is software based.

Considering the throughput requirements of some of the applications in space-craft control it appears that in future distributed computation on concurrent systems with fail-safe and fail-operational features would have considerable attention. With the present trends based on VLSI based hardware technology the importance for methods of task partitioning, distribution, and system reconfiguration in the case of designing fault-tolerant system is our present challenge.

Reconfiguration for distributed fault-tolerance

An analysis and design method for distributed fault-tolerance(DTF) based on graph theoretic results has been reported in [31]. The vertices of a graph represent processing elements and the edges represent channels or direct links between processor. It is assumed that the channels are not sharable.

Although distributed systems have many definitions, most of them possess three common properties, viz. modularity, parallelism and autonomy. The designing goal of the DTF system is to guarantee the correct execution of critical tasks. Due to the modularity and parallelism properties of distributed system, a critical task can be replicated on three similar units called a rep-set, and be processed concurrently by this set. The result obtained by each replicator is transmitted through different paths of the communication network to other replicators. By voting these results the rep-set can obtain the correct answer and locate the possible faulty replicator. Since all the units of the system are autonomous, i.e., there is no central host unit or hard core of the system, each unit having been faulty can be isolated from the system by the other tow normal units of the same rep-set. The DTF system takes care of both processor and channel failures. It assumes total autonomy of the independent processing modules in the system. Each faulty processor can be isolated from the system by the other two normal unit of the same rep-set in an autonomous manner. Thereafter, the two fault-free units select an appropriate unit to form a new rep-set and thus reconfigure the system.

The set of replicators (rep-set) is modeled by three vertices (interchangeably used with processors) of the graph. Since the vertices in the rep-set represent processors that should be able to check the results of the others in the set, it is necessary that they should be connected by paths. The detection and diagnosis of a fault isolates a vertex from this set. This produces a degenerate graph with the vertex deleted.

In order that the system remains a fault-tolerant one need to find a new rep-set in the system. This is the step of reconfiguration and is the essential basis of fail-operational behavior of the system.

The problem of designing a DFT then is equivalent to finding sequence of rep-sets corresponding to the sequence of degenerate graph that follows due to possible failures of vertices (processors) of the original graph. Given a graph topology, along with an initial distribution of the tasks, one problem in designing a distributed fault tolerant system is to find how many successive faults can be tolerated by such a system. In other words, how many successive times will it be possible to have a rep-set in the graph of the system under the possibility of a vertex (failed processor) from the rep-sets being deleted. This is termed as the degree of fault-tolerance of the graph. Based on an analysis to produce a graph topology that is most amenable to providing a high degree of fault-tolerance simple topologies have been suggested.

A relationship between the architecture and the fault-tolerance degree of a system has been established. Experiments were reported on a ring of single-board computers. Similar to the SIFT fault-tolerant computers, all fault-tolerance facilities including detection, diagnosis, isolation, reconfiguration and clock synchronization etc.were implemented by software

Embedding techniques for efficient computation on Hypercubes:

One of the crucial problem of efficiently utilizing a highly distributed system is that of assigning the tasks (load balancing) to achieve optimal resource management. This is called the problem of mapping or embedding.

In a distributed computation a set of tasks are related to each other by the need for communicating for accessing data and code. In a distributed computer where the different tasks communicate over non-shared bus the role of assignment of the tasks affects the cost of such communication. As the cost of communication may become more than that of processing [38] it is necessary that an optimal assignment of the tasks to processors in the system is made.

An hypercube concurrent computer is a distributed system of this type. It is based on the interconnection topology given by the Boolean n-cube[20]. With the development of VLSI technology and the regularity of the n-cube structure the cost of such systems with considerable number of processors have become manageable. In many of the works on hypercube embedding the problem has been treated as that of mapping a graph that corresponds to the application program to the n-cube graph structure of the hypercube[87]. The constrains used in such embedding are the length of the path (dilation) that connects pairs of vertices of the hypercube that are the associated to the end-vertices of an edge of the application graph. The length of the path in the hypercube gives the communication cost associated to the interaction of the two processes located at the two processors corresponding to the end vertices.

Considerable research has been reported [26,27,28,29,30,39,40,43,69, 71,78,79] in hypercube embedding and mapping methods. Significant research is also ongoing in the area of improving the communication and related issues in the hypercube machine[90].

As already mentioned above, the main objective of mapping is load balancing. Several methods for load balancing has been suggested in the literature [100,101]. However, the basic problem of hypercube embedding being NPC [87] researchers have to limit only to obtain near-optimal methods that are acceptable. The main strategy in obtaining such embedding is to obtain a near-optimal load assignment without increasing the over-head considerably. It is possible to have either static (where the structure of computation does not vary very much) mapping or dynamic mapping. Heuristic algorithms for load balancing based on internal structure of the application as well as some other criteria (e.g. measure of load imbalance) have been suggested. Existing reports based on graph theoretic properties of the hypercube is of great importance[87]. An automated problem mapping system called Crystal runtime system has been reported[88]. Several operating system are designed on hypercube that considers the communication topologies of applications (that are currently recorded for program monitoring and exception handling) for use in the implementation of the object model on the hypercube.

Research in embedding methods for the hypercube concurrent computing machines and the growing interest in using distributed fault-tolerant computers for critical applications in space missions raises a natural interest to look into finding ways of devising embedding methods that will provide high degree of fault-tolerance in such systems. The graph based definitions used for designing DFT, mentioned earlier, gives us a way to relate fault-tolerance to different aspects of graph embedding.

Reconfiguration as a graph problem based on a rooted tree

It can now be noted that the requirement of fault-tolerance as in [31] is the capability of building a rep-set with at least 3 vertices. When replicated processes run on 3 processors each checking the other for fault, it is necessary that on the occasion of a fault in one of the processors a different fault-free processor need to replaces it to form a new successor rep-set.

In terms of the graph that models the distributed system this means that in the degenerate graph that has been obtained after deleting the vertex corresponding to the faulty processor one has a vertex other than the remaining two (corresponding to the fault-free ones) that remains connected. The need for having disjoint paths connecting the vertices in the degenerate graph, this demands existence of a cycle containing the three vertices under consideration.

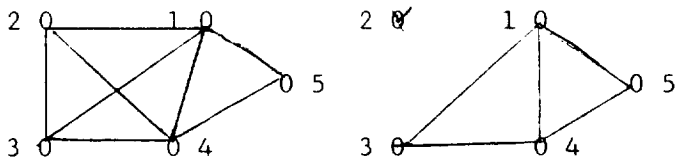
As observed in [85] in case of the regular structure of the hypercube, the other requirements for reconfigurability; that of the ability to reconfigure the system using minimum delay, taking advantage of the

structure of the system, needs to be considered for effective fault-tolerance.

The hypercube graph is a bipartite graph. One can find a maximal bipartite subgraph of the given graph of the distributed system in order to look for an embedding. Efforts to find a maximal bipartite subgraph of any graph is also limited as this problem is also in the NPC class. However, a recent work reports a simple and efficient approximate algorithm for detecting a maximal bipartite subgraph of a graph by a straightforward breadth-first search[93]. In this work it has been shown that a bipartite subgraph of a graph can be obtained by deleting a nearly minimal number of edges of the graph. It has been proved there that the number of edges deleted can not exceed twice the number of edges that is required to get the exact solution subgraph (the subgraph resulting from the minimum number of edge deletion). A breadth-first spanning tree (BFST) has been used to find the nearly maximal bipartite subgraph in the above work. The complexity of obtaining a BFST is only O(e), where e is the number of edges in the graph.

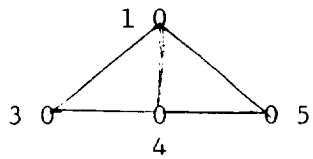
A BFST is a rooted tree where each vertex (including the root) are partitioned into disjoint levels according to its distance from the root. An edge lying between a pair of vertices in the same level represents a chord (called a link-edge as opposed to a tree-edge) of the fundamental cycle with respect to the BFST. The technique of using the BFST and the associated cycles that are fundamental can be used dynamically as the computation proceeds and reconfiguration can be achieved.

As an example consider a simple graph showing the processing nodes and the communication links shown below. It can be seen that a rep-set {1,2,3} can tolerate at least two successive faulty units. This is because it is possible to obtain new (successor rep-sets) rep-sets representing vertices that can test each other using disjoint paths.



When vertex 2 fails a degenerate graph obtained by removing the vertex as shown is obtained. A new rep-set given by {1,3,4}. This can proceed if when a vertex in this new rep-set fails by reconfiguring in a similar manner.

A BFST of the new graph is given below.



The BFST clearly shows the shortest cycles (by definition a BFST gives tree levels that are minimally distant from the root) can be identified. The question of determining a rep-set then is to find a set of vertices that are connected with each other by disjoint paths. As long as there is at least a cycle containing the vertices in a rep-set this process is possible. The existence of a link-edge in the BFST gives a cycle connecting the vertices including the root and all vertices in the path from it to the two end vertices of the link-edge of the graph. One can use the BFST cycles (cycles for each link-edge) for this purpose.

In the above figure, if vertex 3 fails now, one can easily form a successor rep-set by using the cycle corresponding to the link-edge (4,5). The new rep-set is {1,4,5}. Note, if vertex 1 fails then it is not possible to form a successor rep-set because there is no cycle containing the remaining vertices in the old rep-set and a vertex in the graph.

It can be observed that if all the nodes in a level of a BFST fails then it is impossible to construct a new rep-set. By simple techniques a BFST with a certain root can be changed to another with a different root. This gives a method to obtain a layer that has at least some fault-free vertices. The rep-set can then be obtained from the BFST induced graph.

A simple scheme as described below can be used.

Algorithm reconfig

1. input the tasks and mark the critical tasks

2. find a BFST for arbitrary root.

3. compute rep-sets for each of the critical tasks with 3 vertices each. If this is not possible due to nonavailability of cycles then reorient the BFST by using a different root.

4. if a vertex fails then recompute a BFST of the degenerated graph (this can be done from the BFST of the original graph). Compute new rep-sets that are successors of the corresponding old rep-sets.

5. continue step 4 as long as possible. if not possible to obtain a successor rep-set for a failed vertex for the BFST, other BFSTs are constructed and the search is made. when no rep-set can be built then the system is unable to reconfigure.

Relationship of Algorithm reconf to hypercube embedding:

The algorithm **reconfig** that is based on BFST can be used for hypercube load balancing and mapping with reference to reconfiguration for fault-tolerant behavior. The properties of the cycles can be used to obtain new rep-sets by relatively simple algorithms. There is, as already noted earlier, numerous interesting research reports available for mapping and load balancing in the hypercube. It seems there is sufficient results available to support investigations into how reconfiguration can be added onto the process of mapping.

General network topologies and dynamic distribution of processes over a distributed system can therefore be studied in this viewpoint. The subcube to subcube linkage in the case of hypercubes[14,85] and the schemes for designing fault-tolerant hypercube systems based on load-balancing and redistribution can also be studied in this point of view.

Plan of research:

The study of graph embedding methods for distributed computing system (in particular the hypercube concurrent computer systems) and the design of distributed fault-tolerant system have been done. In has been observed that graph theoretic study of reconfigurability gives a way of looking at the problem of building fault-operational system more systematic. At the same it has been seen that the various mapping methods for hypercube might possibly be combined with the requirement of reconfigurability to suggest new integrated ways of tackling the problem.

Breadth-first spanning trees can be used for formulating rep-sets for designing reconfiguration strategies for fault-tolerant systems.

Strategies for reconfiguration in distributed systems will be studies with reference to breadth-first spanning trees and the associated cycles. Graph theoretic methodologies will be studied for obtaining optimal reconfiguration strategies for hypercube interconnection topology.
An embedding program that embeds any distributed computation onto the hypercube structure has been developed. This program uses the BFST based mapping followed by an A* algorithm (written in Interlisp) based on a heuristic defined over Hamming distance will be first converted to Common Lisp and suitably modified to be tested for adaptation to the above reconfiguration scheme.

At the end of the above development the complexity estimation in terms of the real time computing and reliability requirements for spacecraft control computers will be verified and tested.

4.2 Applicative Caching for Avoiding reamputation of Functions

An implementation of applicative caching for stream based computation on a data flow machine suggested by Amamiya has been reported[92]. The construction and maintenance of the directory for caching of functional values have been treated and a caching paradigm suggested.

The problem of eliminating recomputation may help in controlling redundant computation and thereby improve efficiency of computation.

The problem can be illustrated by a function computing Fibonacci numbers as above :

```
fib(n) = if n <=1 then 1
                  else fib(n-1) + fib(n-2) fi
```

As we build the sequence we come to a situation of the form:

```
fib(102) = fib(101) + fib(100)      ..... (1)
```

Here, fib(101) and fib(100) are called in an environment. But

```
fib(101) = fib(100) + fib(99)       ..... (2)
```

Now,

```
fib(102) = [ fib(100) + fib(99) ] + fib(100)
                                          ..... (3)
```

Note, fib(100) appears twice which will go unnoticed during compilation because of static code generation by the compiler. Since the run-time system finds fib(100), written inside the square bracket, in the new environment, it will be recomputed.

While computing fib(100) in (1), if we had some storing mechanism to store the value of fib(100), we do not have to invoke the function for n = 100 again during the evaluation of fib(101). Instead, the value of fib(100) will be read from the store. This reduces enormous computing time and hence results in reduction of resource demand.

The applicative languages (or functional) have the nice property of "referential transparency" meaning that, within the given context, all the multiple occurrences of an expression have the same value and thereby guaranteeing the freedom from side-effects of the expressions. This has the following implications:

a. Singular evaluation and code optimization;
b. Parallel evaluation.

Usually the extent of context is not known at compile time. It develops during the computation process. This leads to different instanciations of

the same expression not being aware of each other's existence as can be seen from equations (1) and (2). Recomputation is avoided by saving associations of argument value and the corresponding function value (i.e. [n,f(n)] if f is a function of a single variable n). Keller and Sleep call such a collection of associations a cache, the function whose values are to be saved a cached function and the associations that are saved the cached values.

Keller and Sleep[56] suggest an applicative implementation by defining an applicative definition of a data structure called directory that contains every cached value that could conceivably be requested. Their computational model is demand driven as it makes it possible to define a directory out of data constructors without causing a priori evaluation of the entire structure. Through the use of a construct called "suicidal suspensions" an application is evaluated when it is first demanded. After this the expression is effectively replaced by its value. Stream directory and Array directory are explained in detail. The special features of applicative caching are (a) allowing the programmer to specify that the recomputation of a function is to be avoided; (b) permitting selective control of the amount of recomputation and programmer's control over purging (the programmer can give a guess value of the reference count); (c) providing a set of applicative building block; (d) even if the value of the function is removed from the cache, it is not fatal--- the value is recomputed.

Keller's work deals with the abstraction of caching at the functional language level, hence has a sound denotational semantics. He also gives caching pragmas which relieves the programmer of the burden of coding proper caching functionals. The compiler translates the pragmas into proper functionals.

In [3] special operators had been introduced to accomplish caching in Amamiya's dataflow machine.

The following special operators have been used to accomplish caching. tcons(n,y,r) : This has three arguments. tcons negotiates with the resource manager and gets three nodes, sets reference counts to one in all the nodes, writes the attribute field of the car of the first node with repetition could of the function value (given by the programmer), car of the first node with the argument n of the function, the car of the second node with the value y=f(n) computed leniently and the car of the third node with zero and returns the address of the list. The remaining fields are filled as given in [56]. This operator has a side effect of allocating a tnode[11]. decr rep(cf) : This is an operator when called into action decrements the attribute field of the car of the first tnode. ccopy(t,cf) : This macro operator has two inputs and one output and is designed using a machine instruction copy.

The semantics of the copy operator is as follows.

```
copy(t,undefined,cf') = cf'
copy(t,cf,cf') = cf
```

where cf is the address of the new cache, cf' is the address of the old cache and t is the trigger signal.

copy is strict in its first argument t.

An operator -- fby, which is a nonpointwise[11] operator with the following semantics:

fby(x0,x1,....,xn,..),(y0,y1,..,ym,..)) =(x0,y0,y1,y2,....)

The output of ccopy is triplicated and sent to the function body, select operator and Cache Handler respectively.

select(cf,cf') : This is a two input operator. The first input is the output of the ccopy and the second is supplied by the function body. The semantics of this operator is as follows:

select(undefined, cf') = cf'
select(cf,cf') = cf

set zero : This is an operator that sets the reference count to zero.

The operators copy, select, decr_rep and set_zero are to be introduced into Amamiya's architecture. We have included these to construct and manipulate the data structure for cache. These are semantically sound in that these are the additional operations we are including in the continuous algebra E(A)[11].

The caching mechanism works by using the inherent list processing capabilities of the dataflow machine and a VALID like language has been used to write the programs. For uniformity of the structure of the cache the directory header is also a tnode with a signature field. It can be noted that for the header, n and f(n) remain undefined.

A LISP based simulator has been written to estimate performance figures of the system. Garbage collection in the system seems to be a crucial task. The problem is all the more complicated because of the need to construct an altogether new cache, whenever there is an appending of a new list z containing the value [n,f(n)] to the old cache.

In Amamiya's proposal once the reference count of a node is reduced to zero it will be garbage collected. However, a number of problems arise while caching because several caches will be generated during a computation and it is necessary that an appropriate memory reclamation should be made for a bounded use of the structure memory. The bound in this context is determined by the user supplied value of the repetition count. This count will help us in maintaining a cache of minimum size. This repetition count will be reduced by one every time there is an access to a value in the cache. To reduce the complexity we use the following technique.

Whenever there is a call to the Cache_Handler (while appending a new list) the cache handler will first check whether the repetition count of a tnode

of an old cache has become zero. If the repetition count has become zero then the cache handler sets the reference counts of all the three cons nodes used in the tnode to zero. The underlying system garbage collector will reclaim these nodes in due course. If the repetition count is not zero then the cache handler gets a new tnode and writes signature 's' which is the number of tnodes deleted after the tnode in question, in the signature field of the corresponding new tnode leniently. It copies the remaining fields of the old tcons node into the new one.

Search is done reccursively over the cache in a linear pipelined fashion. But a problem arises if there is an invocation of cache handler during a search, i.e. when a new cache is being created while some search is pending on the old cache. Also, as the search is invoked reccursively, we must have a mechanism to switch from old cache to the new one when such a new cache is generated. We resolve this by introducing the operator select, which outputs either (1)the address of the part of the old cache in the absence of a new one, or (2)the address of a new cache, if a new cache has been generated.

About the problem of the searches on old caches the following may be noted. If argument expression of such a search is already evaluated completely then the old cache is used. This means a part of the cache is not garbage collected while a new cache is being generated. This is because the repetition count of the tnodes of that part of the old cache is nonzero. The new directory may have been defined by the cache handler before the complete evaluation of the argument of the search. In that case the tcdr macro operator will use the signature field of the tcons nodes and the proper reference is made in the cache automatically.

The part of the old directory which has not been garbage collected remains a burden to the system. We suggest that a signal may be sent to a secondary process which scans this part of the old directory checking the repetition count for zero and setting reference count properly for garbage collection as soon as search completes. This ensures that no unused nodes remain in the system.

Plan of research:

The aspects of applicative caching programming paradigm has interesting applications in digital signal processing, image processing and numerical computing where many recurrence computations are involved. However, it has been observed that in dataflow type machines, if the rate of input stream is high, then there will be too many caches being constructed. This is not desirable. A check on the stream of argument values by a mechanism similar to that of throttling[9,10] will be useful. This ensures that only a few caches will be present at a point in time and the garbage collection overhead may be manageable. The garbage collection complexity for caching should certainly be less than the recomputation complexity in order that this is useful. Our feeling is that saving of resources will be more than offset when compared to the resource requirement of garbage collection.

It has been observed that the possibility of avoiding recomputation in a distributed system leads to lesser number of processes and less probability of deadlock. It however does this at the expense of creating processes that does cache management. However, code sharing and dictionary sharing can reduce this overhead. It is necessary to verify the performance for applicative caching in distributed systems. It is interesting to conduct experimental studies to study the efficiency of caching in distributed systems.

A simulator (written in Interlisp) has been developed for simulating the Amamiya's dataflow architecture and implementing the applicative caching. It will be changed over to Common Lisp and will be suitably modified to take into account performance attributes important for distributed computing with particular reference to space mission applications.

4.3 Software fault-tolerance involving design diversity and N-version programming

Introduction

Software Fault Tolerance is an area of study which began to receive real attention in the mid to late 70's. The number of researchers active in this field, has been and still is, relatively low, although there has been a flurry of research activity in the past few years.

Early research projects attempted to mirror engineering's use of redundant hardware modules to ensure fault tolerance. The fallacy of using redundant software modules is that, hardware and software component failures are not at all similar. Hardware errors are not usually coincident (errors which occur at the same time), whereas many software errors are [36,37]. Also identical software units will produce identical errors, this is not normally true for hardware components. There were, and are, established methods of assuring hardware reliability; metrics to assure software reliability were, and still are, abysmally few.

Taxonomy: Design Diversity

Later, and much of today's, research attempted to adjust for coincident errors through the use of design diversity and/or N-version programming [4,5,6,13,18,91]. N (N > 1) designers receive the same software requirements specification and must produce a detailed system design. The designers are expected to perform their task while in complete isolation from the other system designers. The detailed designs are then forwarded to the programmers. The end products (software modules) should be equivalent. N-version programming is the production of multiple software modules, each module having been independently programmed using the same design. Practically, very little distinction is made between design diversity and N-version programming. In most of the literature these terms are interchangeable.

After these modules have been produced and independently tested they are assembled into a system and run concurrently. At selected inspection points, during execution, results are collected and evaluated. The results are expected to be "similar". Any "non-similar" results are rejected.

The design diversity paradigm [58,59] assumes that faults exhibited by individual modules will be non-coincident. Different (but equivalent) designs will produce software modules which fail non-coincidentally (none of the N modules will produce the same error at the same time).

Results: Design Diversity

Almost all of the research activities involving these methods have produced prototypes which exhibit high degrees of fault tolerance. One paper produced results which are critical of this paradigm. The primary criticism is that the assumption of independence is not always valid. Nevertheless, where this assumption of independence is valid N-version programming is a superior fault tolerance paradigm.

Problems: Design Diversity

The most significant problem I find with this methodology is that humans are responsible for producing the modules. We have similar training, use similar equipment, therefore we will produce similar errors in our designs and code. This fact tends to weaken any assumption of independence, which weakens the argument for design diversity.

Another problem is the assumption of equivalence of the different designs. This is probably why most of the experiments really employ N-version programming instead of design diversity. The entire requirements-to-detailed design phase needs to automated; and to really produce diverse designs the automation tools should be different. After the detailed designs have been created, another automated tool must be used to determined to prove equivalence. Humans can not perform this task! Once the designs have been proven to be diverse yet functionally equivalent, then the code can be produced by programmers. The resulting modules should be truly diverse. Present research attempts to create diversity by coding the separate in different languages, or using different non-automated methods to specify the problem, these are not sufficient.

The most important papers found in the reference materials are by Eckhardt and Lee. They focus on the coincident error problem and attempt to produce a metric by which the probability of the numbers of coincident errors can be predicted. Using this metric, one can determine when the use of multiple modules (created using design diversity or N-version programming) is preferable to the use of a single software module.

Taxonomy: Others

Other taxonomies (these need further study) are Exception handling [33], recovery blocks [80], and synchronization graphing [6]. None of these methods seems to be as promising as design diversity, however, these

research areas may provide constructs which can alleviate the independence problem.

Very little research has been expended in these areas. I feel that these are directions which should be investigated further.

Future Directions

Fault tolerance is an area of research which has seen few real advances in the past ten years. Most efforts are channeled into one of the theories described above. There may not, in fact, be a better method to assure software fault tolerance than through design diversity or some combination of all of these methods. There should at least be a effort to find one, if it exists. I propose that some time be expended to explore alternatives to design diversity. This effort need not have high priority, but this type of investigation needs to continue.

Since design diversity/N-version programming seem to be the method of choice, an major effort needs to be made to "pull all of the pieces together". A review of the pertinent literature reveals that there in a majority of projects there seems to be a needless duplication of effort. A short term study should be made with the sole purpose of illuminating the real differences in past research efforts and explaining how these differences effect the issue.

Particularly disappointing is the general strategy most researchers have been using to produce fault tolerant systems. There seems to be an unspoken consensus that true design diversity (the use of different but equivalent designs) is extremely difficult to achieve given today's technology. Therefore, N-version programming is utilized in the preponderance of experiments. N-version programming is not the answer, we must explore means to easily achieve and implement design diversity. In the short term a taxonomy for the implementation of the "front end" of design diversity should be investigated. Can some N versions of automated design tools be created which will produce equivalent designs? More importantly, can an automated equivalence prover be implemented? If the process can not be automated, at this time (or ever), what manual methods could be performed to replace automation? What is an optimum value for N? Supposedly, 3 is too small but anything greater than 5 may be prohibitively expensive (these numbers are used merely to illustrate the argument no research has gone into determining whether of not they are valid limits) and unnecessary. These questions need to be answered.

4.4 Distributed neural computations on parallel systems

Background:

Neurocomputing (also called neural networks, connectionist systems, artificial neural systems, associative models of computation, etc.) has received a great deal of attention in the technical literature. This field

is one of the fastest growing and most innovative areas of computing, and have even been called "more important than the atomic bomb". However, due to rapid growth and proliferation of publications, it is difficult to provide a clear concept of what the field is all about and put it to practical use. Generally, the neural networks have potential to solve complex non-deterministic problems such as speech and pattern recognition at high speeds.

A brief overview and taxonomy can be found in [96], where McCulloough-Pitts neuron model, perceptions (and their limitations that have retarded the growth of this field for two decades), Hopfield Nets, and the current models (Hinton's Bolzman Machine, Rumelhart's back propagation learning rule, etc.) are mentioned. A more comprehensive review of literature and recent developments in this field can be found in **NASA JSC Neural Network Survey Results** [21]. Since the field is rapidly changing, in addition to 142 references cited in the NASA Survey, we have compiled an even more up-to-date bibliography that list books and articles separately in Appendix II.

In order to gain insight into this new field, we have compiled Neural Network Primer series Part I through Part VII [21] in a binder and made it available to the interested minority faculty and students including the research assistants for the PAPER project. The primer series covers a wide range of learning systems from very simple bin sorter to the more advanced unsupervised systems based on drive-reinforcement theory [57].

Multiple Neural Components Working on Subtasks

Our research interest in this area is to put the theoretical results of neural computing to practical use, and as such, can be characterized as applied. As was discussed in Phase 1 proposal, we would like to add neural computation capability to multicomputer systems that otherwise support the intended conventional supercomputing functions.

We intend to do this for large-scale neural computations with unacceptably long learning times. The neural computing capability for a parallel system will be realized at the architectural level by addition of neural components (either implemented in VLSI boards or simulated by software) to the processing nodes. On the side of the application, large-scale neural computations such as training methods for cognition problems that involve a large number of complicated categories will need to be decomposed into subtasks. The decomposition is made in such a way as to allow uniform and quick learning periods with rapid convergence for each subtask. The overall computation will then involve the learning subtasks assigned to the processing nodes that implement the neural sub-networks.

As is discussed in [52], a problem such as recognition of Japanese Kanji character set that has almost 3000 categories (2970 categories) may require unacceptably long times with too many learning cycles for convergence This is due to uneven learning for "simple" and "complicated" categories, and the tendency of the large neural net to first learn the easy categories before trying to learn the difficult ones. The problem facing the constructions of

large scale networks is that the requirements for computational resources increase as the number of categories increase. Thus a monolithic network on recognizing one thousand or more categories would soon run up against the limitations of currently available compute resources. Furthermore, the networks recognition performance is poorer for large number of categories [52]. This suggests that a large-scale neural network for character categorization should be divided into independent subnetworks, each of which focuses on appropriate sub-set of characters.

Learning Methods for Large-Scale Hierarchial Neural Networks

After the problem is decomposed, simple application of backpropagation methods on multi-layered neural networks will not be sufficient. Novel learning methods, such as those given in [52], are required for neural computations that are broken up into sub-networks. Finally, once the subtasks are learned in reasonable periods, there is a need for appropriate interation.

Implementation on Parallel Machines

There are various decomposition methods for implementing neural networks on parallel machines. An example involving the decomposition by samples is given in the above-mentioned article. There, 7100 input samples (71 categories times 100 sample per category) for Hirangana set have been divided into 2^n subsets and run on a hypercube machine. However, the easiest and most effective way to decompose a large-scale hierarchial network is to implement the sub-networks in parallel. For example, if a large scale hierarchial neural network consists of M sub-networks and a super-network (used to integrate the sub-results), the sub-networks can work simultaneously.

Plan of Research

As discussed, a large-scale neural learning task can be broken up into autonomous subtasks, where some care needs to be taken in devising balanced learning periods for each subtask. The sub-networks can then simultaneously work on the subtasks. We feel that, regardless of the specific learning method (e.g. back-propagation drive-reinforcement, etc.), this is an ideal application for a parallel computing where the processing nodes contain entire neural networks.

Thus, our first task is to find a way to add neural computing capability to the processing nodes of a parallel machine that is PAPER class, meaning that it is particularly suitable for computational support of a, say, planetary rover. This can be done by inclusion of neural computing boards, such commercial product offerings as HNC Anza+, Anza 2500, or TRW Mark-III. These neural components are offered as extensions 32-bit high-end PCs [8], and there are various other VLSI implementations for Von Neumann type processing nodes [9,10,11,12].

As was mentioned in Section 3.2, further enhancement of the MAX Module shown in Figure 3 with neurocomputing capability is important for demanding

cognition functions that must be performed in a planetary exploration mission. This can be done through addition of neural VLSI components to the nodes. We do not envision any obstacle, since this will be accomplished in a way similar to the addition of vector or floating point extensions to a commercial parallel machine at processing node level [51].

We would like to collaborate with research staff at JPL/Caltech involved in MAX project in order t first conduct a system-level feasibility study, and subsequently investigate the design issues. After neural computing capability is added to a parallel system such as MAX, the next step will be construction of appropriate learning methods such as those in [54]. Finally, additional research needs to be done on identifications of large-scale neural computations, and decomposition of the computational supertask into parallel subtasks.

V. PROPOSED FUTURE ACTIVITIES

We have plans to pursue the examination of the two architectures, viz. AIPS, CSDL and the MAX, JPL/Caltech. For this we have plans to visit JPL/Caltech in July followed by a visit to CSDL in next Fall. We shall also start our study of similar work, like VHSIC multiprocessor being developed at NASA/LaRC and the CHRP system of The Goddard Space Flight Center.

5.1 Trip to Jet Propulsion Laboratory

We have contacted JPL/Caltech Historically Black Colleges and Universities (HBCU) office for arranging a visit to get direct information regarding experimental computer architectures for NASA's future space missions. In addition, we have contacted JPL research staff (Dave Eisenman, David A. Nichols) to obtain preliminary information on experimental architectures. Our conversations with staff involved in the MAX project and Mars Rover project were useful and productive.

The co-investigators plan a three-day visit to JPL/Caltech some time in July 15 time frame. The purpose of the trip will be to meet with key research staff in order to get direct information regarding:

- Experimental architectures such as MAX, Snooping-Cache Multiprocessor, JPL-Star, Mark III Hypercube, etc.
- Collect requirements information on Mars Rover project
- Gather fault tolerance requirements data for space missions
- Obtain direct information on software fault tolerance

5.2. Trip to Charles Stark Draper Laboratory

We have met with Jay Lala of Charles Stark Draper Laboratory during the CSDL Program Presentation at NASA Langley Research Center on March 1,2. At this meeting, we have obtained the relevant material on AIPS studies, including the AIPS System Requirements and AIPS Technology Surveys.

We plan a visit to CSDL sometime in Fall 1989 in order to obtain more up to date information regarding the distributed software for AIPS Proof of Concept Configuration. This work will be nearly complete at that time, and may provide useful information on distributed software for fault tolerant systems. Our main contact person for this information is Linda Alger.

In addition, Dr. Ranjan Sen will obtain more detailed information that is relevant to his project on "Reconfiguration for Distributed Fault Tolerance" from the work carried out in regards to intercomputer communication user services. The main contact in this area is Laura Burkhardt.

5.3. Attendance at HBCU Symposium

In addition to the trips above, we plan to attend the HBCU Symposium to be held in Atlanta, Georgia in Fall 1989. This will be a three-day trip, and will also include the student research assistants.

VI. CONCLUSION

In this report we have looked at the requirements for space missions as given in the somewhat dated AIPS System Requirements document and contrasted those with the new requirements from JPL/Caltech involving sensor data capture and scene analysis. It shows that the requirements change with the changing technology.

Additional work needs to be done in the remainder of Phase 1 and in subsequent phases of the project in order to gather more detailed requirements data, extract the components relevant to planetary probes and organize these in a way to ascertain and establish the required architectural features of computing systems.

Presently, we have evaluated two possible architectures, namely AIPS POC Configuration and MAX Fault-tolerant Dataflow Multiprocessor. The main observation was that the AIPS Design is biased towards fault tolerance and may not be an ideal architecture for planetary and deep space probes due to high cost and complexity. The MAX concepts appears to be a promising candidate, except that detailed information is required. We intend to pursue the evaluation of MAX with a visit to JPL/Caltech and explore the feasibility of research for adding neural computation capability to this architecture.

We have identified key impact issues for architectural design of computing systems meant for planetary missions. We have described these in detail as focus areas. We would like to conduct research in the aforementioned focus areas. We shall also continue to use the funding for Phase 1 to provide research experiences for minority faculty and students.

A few areas of research which we feel can be taken up in subsequent phases of the project are:

- Reliability models, similar to in AIPS Study, using probabilistic Markov chain analysis. The models need to be far simpler taking into account fewer factors and assumptions and should be able to predict reliabilities with or without any fault-tolerance.
- Architectural tradeoff studies, especially in regards to the degree of fault tolerance, and the associated overhead.
- Checkpointing and rollback techniques as recovery methods from faults in the absence of software or hardware redundancies.
- Uniform approaches to hardware and software fault tolerance.

VII. REFERENCES

1. AIPS System Requirements, NASA Report No. AIPS-83-50, Charles Stark Draper Laboratory, Cambridge, Ma., August 30, 1983.

2. AIPS Technology Survey Report, NASA Report No. CSDL-C-5691, Document No. 43-315, February 1984.

3. Amamiya,M,Hasegawa,R,Nakamura,O and Mikami,H "List Processing Oriented Dataflow Machine" Proc.NCC, Vol.No.157,AFIPS 1982.

4. Anderson, T and J.C. Knight, A Framework for Software Fault Tolerance in Real-Time Systems, IEEE Trans Software Eng., Vol. SE-9, No. 3, May 1983, pp 355-364

5. Anderson,T, P.A. Barrett, D.N. Halliwell, and M.R. Moulding, Software Fault Tolerance: An Evaluation, IEEE Transactions of Software Eng., Vol SE-11(12), Dec 1985, pp 1502-1510.

6. Anderson,T, P.A. Barrett, D.N. Halliwell, M.R. Moulding, An Evaluation of Software Fault Tolerance in a Practical System, 15th IEEE Int. Symp. Fault-Tolerant Computing, 1985, pp 140-145.

7. Andre F, et al, "Experiments with mapping algorithms on a hypercube", 4th Hypercube Conf. Monterey, CA, Mar 1989.

8. Anza-Plus turns PC-AT into a Neurocomputer, Computer,Vol. 21, No. 3 (March 1988), pp. 134.

9. Arvind and Culler,D.E. "Managing resources in parallel machines",Fifth Gen. Comp. Arch., Woods,(ed),Elsevier Sci.Pub., North-Holland IFIP 1986.

10. Arvind and Nikhil,R.S.,"Executing a program on MIT tagged token dataflow architecture", Comp.Struc.Grp. Memo.271, Mar 1987.

11. Ashcroft,J "Eazyflow", LNCS 224, pp 1-50, Springer-Verlag 1985.

12. Avizienis, A, J.P. Kelly, Fault Tolerance by Design Diversity: Concepts and Experiments, IEEE, 1984.

13. Avizienis,A., M.R. Lyu, and W Schutz, In Search of Effective Diversity: A Six-Language Study of Fault-Tolerant Flight Control Software, IEEE, 1988.

14. Banerjee, P, "Reconfiguring a hypercube multiprocessor in the presecne of faults", 4th Hypercube Conference, Monterey, CA, 1989.

15. Banerjee, P, "Reconfigurable cube-connected cycles architectures", FTCS 1986, pp 286-291.

16. Banerjee, P et al, "An Evaluation of System-level Fault Tolerance on the Hypercube Multiprocessor", FTCS 1988, pp 362-367.

17. Berman, F et al "On mapping parallel algorithms into parallel architectures", Journal of Parallel and Distributed Computing, 4,431-458(1987).

18. Bishop,P.G., D.G. Esp, M. Barnes, P Humphreys, G. Dahll, and J. Lahti, PODS - A Project of Diverse Software, IEEE Trans. Software Eng., Vol SE-12(9), Sept 1986, pp 929-940.

19. Bollobas, Bela, "Extremal graph theory", Academic Press, 1978.

20. Bondy J.A. and U.S.R. Murty, "Graph Theory with Applications", Americal Elsevier, 1976.

21. Caudill, M., "Neural Networks Primer -- Part I thru Part VII", AI-Expert, Dec, 1987; Feb/Jun/Aug/Nov, 1988; and Feb/May 1988.

22. Cezzar, R. et. al. "Parallel Architectures for Planetary Exploration Requirements," NASA Project Grant No. NAG-1-949, 1989.

23. Cezzar, R. nad Klappholz D., "Process Management Overhead in A Speed-up Oriented MIMD System," Proc. of 1983 Int'l Conf. on Par. Proc., August 23-26, 1983.

24. Cezzar, R. "Simulation of Process Management Policies on an MIMD System," Paper abstract submitted to Frontiers '88, Fairfax,Virginia, October 10-12, 1988.

25. Chau, S. N. et. al. "Fault-Tolerance Requirements for High Performance Spaceborne Parallel Computers," JPL Task Plan 81-2984, March 31,1989.

26. Chen, Bethany M.Y, "Dilation-2 embeddings of grids into hypercube", Int. Conf. on Parallel Processing, August, 1988.

27. Chen, Bethany M.Y, "Embedding of 3-dimensional grids into hypercubes", 4th Hypercube Conf. Monterey, CA, Mar 1989.

28. Chen, Bethany M.Y, "Embedding of d-dimensional grids into optimal hypercubes", Symp. Parallel Algorithms and Architectures, June 1989.

29. Chen, Ming-Syan et al, "Embedding of Interacting Task Modules into a Hypercube", Hypercube Multiprocessors 1987, SIAM Philadelphia 1987.

30. Chen, S.K. et al "An Efficient Multi-Dimensional Grids Reconfiguration Algorithm on Hypercubes", FTCS 1988, pp 368-373.

31. Chen, Y et al , "DFT: Distributed Fault Tolerance - Analysis and Design", pp 280-285.

32. Chor Benny and Brian A. Coan, "A simple and efficient randomized byzantine agreement algorithm", IEEE Tran. Software Engg. SE-11, 6, June 1985.

33. Cristian,F Exception Handling and Software Fault Tolerance, IEEE Trans. Computers, Vol. C-31, No. 6, June 1982, pp 531-539.

34. CSDL Completion of AIPS Tasks (FY 88 Tasks), Program presentation at NASA LaRC Building 1202, Hampton, Va., March 1-2, 1989.

35. DeVegvar, P.G. and Graf, H. P., "Studies of Associative Memory and Sequence Analyzer Circuits on a Programmable Neural Network Chip,"(to be published, the author may be contacted for preprint).

36. Eckhardt, D.E and L.D. Lee, Fundamental Differences in the Reliability of N-Modular Redundancy and N-Version Programming, The Journal of Systems and Software, Vol 8, 1988, pp 313-318.

37. Eckhardt,D.E. and L.D. Lee, A Theoretical Basis for the Analysis of Multiversion Software Subject to Coincident Errors, IEEE Trans. Software Eng., Vol Se-11(12), Dec 1985, pp 1511-1517.

38. Fox, G.C. and J.G. Koller, "A Dynamic load Balancer on the Intel Hypercube", #rd Conf. Hypercube Concurrent Computers and Applications, Pasadena, CA, Jan 1988.

39. Fox, G.C. et al, "The Implementation of a dynamic load balancer", Hypercube Multiprocessors 1987, SIAM Philadelphia 1987.

40. Ghafoor, A et al, "An interconnection topology for fault-tolerant Multiprocessor system", Supercomputing 88, ppl36-143.

41. Gilley, G. "The fault-tolerant spaceborne computer (FTSC), Americal Astronautical Society AAS 79-615, Feb 24-28, 1979, Keystone, Colorado, Annual Rocky Mountain Guidance & control Conference.

42. Gluch David P, Paul M.J., "Fault-tolerance in distributed digital fly-by-wire flight control systems", IEEE, CH2359-8/86/0000-0507 , 1986.

43. Graf, H. P. et. al. "VLSI Implementation of Neural Network Model," Computer, Vol. 21, No. 3 (March 1988).

44. Graf, H.P. and deVegvar, P.G. "A CMOS Implementation of a Neural Network Model," Advanced Research in VLSI, MIT Press, Cambridge, Ma., c. 1987, pp. 381-385.

45. Greenwood, D., "NASA JSC Neural Network Survey Results," First Annual Workshop on Space Automation and Robotics, pp. 97-110, August 1987.

46. Gustafson, John L, et al "Development of parallel methods for a 1024-processor hypercube", SIAM Journal on Sci and Stat Computing, 9, 4, July 1988, pp 609-638.

47. Harper, Richard et al, "Fault Tolerant Parallel Processor Architecture Overview", FTCS 1988.

48. Hartmann Robert L, "Interconnection Topology and the Active Graph Parallel Processor", 20 Hawaii Int. Conf. System Sciences, 1987.

49. Horvath, J. C. "Spacecraft Sequencing on the Hypercube Concurrent Processor," HCCA4, Monterey, Califonia, March 6-8, 1989.

50. INTEL, iPSC/2 Simulator, Release 2.1, Intel Corporation, April 1989.

51. iPSC/2, A Product Summary by Intel Scientific Computers, 15201 N.W.Greenbriar Pkwy, Beaverton, Or. 97006.

52. Joe, et al. "Simulation of a Large-Scale Neural Network on a Parallel Computer," Character recognition by subtasks.

53. Johnsson S. Lennart, "Communication Efficient Basic Linear algebra computations on hypercube architectures", Journal of Parallel and Distributed Processing, 4, 133.172 (1987).

54. Jones, M. A. , "Programming Connectionist Architectures," AT&T Technical Journal (January/February 1988).

55. JPL 85 "Hypercube Research Project Mark III Core Engineering Notebook," Report # JPL D-2431, Jet Propulsion Laboratory, Pasadena, CA June 3, 1985.

56. Keller,R.M.and Sleep,M.R. "Applicative caching", ACM TOPLAS, Vol.8,No.1, Jan 1986.

57. Klopf, A. H., Drive-Reinforcement Learning: A Real-Time Learning Mechanism for Unsupervised Learning, IEEE First Annual Int'l Conf. on Neural Networks, San Diego, Ca., 21-24 June 1987.

58. Knight, J.C, N.G. Leveson, L.D. St.Jean, A Large Scale Experiment in N-Version Programming, 15th IEEE Int. Symp. Fault-Tolerant Computing, 1985. pp135-139.

59. KnightJ.C. and N.G. Leveson, An Experimental Evaluation of the Assumption of Independence in Multiversion Programming, IEEE Trans. Software Eng., Vol SE 12(1), Jan 1986, pp 96-109.

60. Kramer O and H. Muhlenbein, "Mapping strategies in message-based multiprocessor systems", Parallel Computing 9(1988) 213-225.

61. Lala, J. H., "Hardware and Software Fault Tolerance: A Unified Architectural Approach," FTCS-88, June 27-30, 1988, Tokyo, Japan.

62. Lala, J. H., "Advanced Information Processing System," AIAA/ IEEE Digital Avionics Conference, Baltimore, Md., December 3-6, 1984.

63. Lala, J "A Bizantine Resilient Fault Tolerant Computer for Nuclear Power Plant Applications" , FTCS 1986, pp 338-343.

64. Lambert, K. et. al., "Planetary Rover Computational and Data Storage Requirements, Version 1.1.," Draft Report, JPL, 4/6/89.

65. Laprie, J., "Dependable Computing and Fault Tolerance: Concepts and Terminology," FTCS-85, pp. 2-11, 1985.

66. Lee, R. A., Session Chair for Fault-Tolerance Mini-Symposium, HCCA4, Monterey, California, March 6-8, 1989.

67. Lee Soo-young et al, "A mapping strategy for parallel processing", IEEE Trans Comp. C-36, 4, April, 1987.

68. Lewnes, A. "INTEL's i860 Microprocessor: Putting Out-of-This-World Performance Within Reach," Microcomputer Solutions, A Publication of Intel Corporation, (May/June 1989).

69. Li C.C. and W.K.Fuchs, "Graceful degradation on hypercube multiprocessors using data redistribution," Int. Conf. Parallel Processing, Aug 1989 (submitted).

70. Littlewood,B, Theories of Software Reliability: How good are they and how can they be improved, IEEE Trans. Software Eng., Vol SE -6, 1980, pp 489-500

71. Livingston, M. and Quentin F. Stout, "Parallel allocation algorithms for hypercube and meshes", 4th Hypercube Conf. Monterey, CA, Mar 1989.

72. Maehle, E et al, "A graph model for diagnosis and reconfiguration and its application to a fault tolerant multiprocessor system",FTCS 1986, pp 292-297.

73. MIL 217D Military Handbook, "Reliability Prediction of Electronic Equipment", MIL-HDBK-217D, DoD, Washington, D.C. Jan 1982.

74. Miller, Russ and Quentin F. Stout, "Some graph- and image-processing algorithms for the hypercube, Hypercube Multiprocessors 1987, SIAM Philadelphia 1987.

75. Nagel,P.M. and J.A. Skrivan, Software Reliability: Repetitive Run Experimentation and Modeling, NASA Rep. CR-165036, 1982.

76. Nagel,P.M., F.W. Scholz, and J.A. Skrivan, Software reliability: Additional investigations into modeling with replicate experiments, NASA Rep CR-172378, 1984.

77. Parberry Ian, "Parallel Complexity Theory", Research Notes in Theoretical Computer Science, John Wiley and Pitman, 1987.

78. Plaxton, C.G, "Load Balancing, Selection and Sorting on the hypercube", 4th Hypercube Conf. Monterey, CA, Mar 1989.

79. Quentin F.S and B. Wagar, "Passing massages in link-based hypercubes", 3 Conf. on hypercube multiprocessors, Pasadena,CA 1987.

80. Randell,B System Structure For Software Fault Tolerance, IEEE Trans Software Eng, Vol SE-1, June 1975, pp 220-232.

81. Rasmussen, R. D., "MAX Fault Tolerance Slide Presentation," April 27, 1989, JPL, Pasadena, California.

82. Recent AIPS Technology Survey Review, Program presentation at NASA LaRC Building 1202, Hampton, Va., March 1-2, 1989.

83. Rennels, D. A. et. al. "Fault Tolerance in A Large Snooping-Cache Multiprocessor," JPL Task Plan 81-2590, March 31, 1989.

84. Rennels, D. A. "Architectures for Fault-Tolerant Spacecraft Computers," Proc. IEEE, Vol. 66, No. 10 (October 1978).

85. Rennels David A, "On Implementing Fault-Tolerance in Binary Hypercubes", FTCS 86, 1986.

86. Rennels, D. A. "On implementing fault-tolerance in binary hypercubes", FTCS 1986, pp 344-349.

87. Saad,Y and M.H. Schultz, "Topological properties of Hypercubes", Research Report YALEU/DCS/RR-389, Yale University, June 1985.

88. Saltz Joel H et al, "Automated Problem Mapping : The Crystal Runtime System", Hypercube Multiprocessors 1987, SIAM Philadelphia 1987.

89. Sammur N.M et al, "Mapping signal processing algorithms on the hypercube", 4th Hypercube Conf. Monterey, CA, Mar 1989.

90. Schwan Karsten et al, "Mapping parallel applications to a hypercube", Hypercube Multiprocessors 1987, SIAM Philadelphia 1987.

91. Scott, R.K., J.W. Gualt, D.F. McAllister, and J. Wiggs, Experimental Validation of Six Fault-Tolerant Software Reliability Models, Proc. IEEE Conf. Fault-Tolerant Computing, 1984, pp 102-107.

92. Sen, Ranjan K, "On the implementation of caching in a Data Flow Computer", ICCI'89, Computing and Information, Vol.II, Canadian Scholar's Press Inc., Toronto, 1989.

93. Sen, Ranjan K, "An approximate algorithm for minimum edge deletion bipartite subgraph problem", 20 Southeastern Conf. Combinatorics Grpah Theory Computing, Boca Rotan, Florida, Feb 1989.

94. Sivilotti, M.A., et. al. "VLSI Architectures for Implementation of Neural Networks," Proc. Conf. Neural Networks for Computing,Denker, J. S. ed. 1986, American Institute of Physics Conf. Proc.151, pp. 408-413.

95. Sklaroff,J.R., "Redundancy management technique for space shuttle computers", IBM J. Res. Develop., Jan 1976.

96. Sondak, N. E. and Sondak V. K., "Neural Networks and Artificial Intelligence," ACM SIGCSE Bulletin, Vol. 21, No. 1, February 1989,pp. 241-245.

97. Stenstrom P, "Reducing Contention in Shared-Memory Multiprocessors", IEEE Computer, Nov 1988.

98. Tombolian, S. "Introduction to a System for Implementing Neural Net Connections on SIMD Architectures," ICASE Report No. 88-3, NASA Contractor Report 181612.

99. Wensley John H, et al, "SIFT: Design and analysis of a fault-tolerant computer for aircraft control", Proc. IEEE, 66, 10, Oct, 1978.

100. Williams, Winifred I, "Load balancing and Hypercubes", Hypercube Multiprocessors 1987, SIAM Philadelphia 1987.

101. Zubair M and S.N. Gupta, "Embeddings on a Boolean Cube", Col. presentation, Old Dominion University, Norfolk, Virginia, January 19, 1989.

VIII. FIGURES

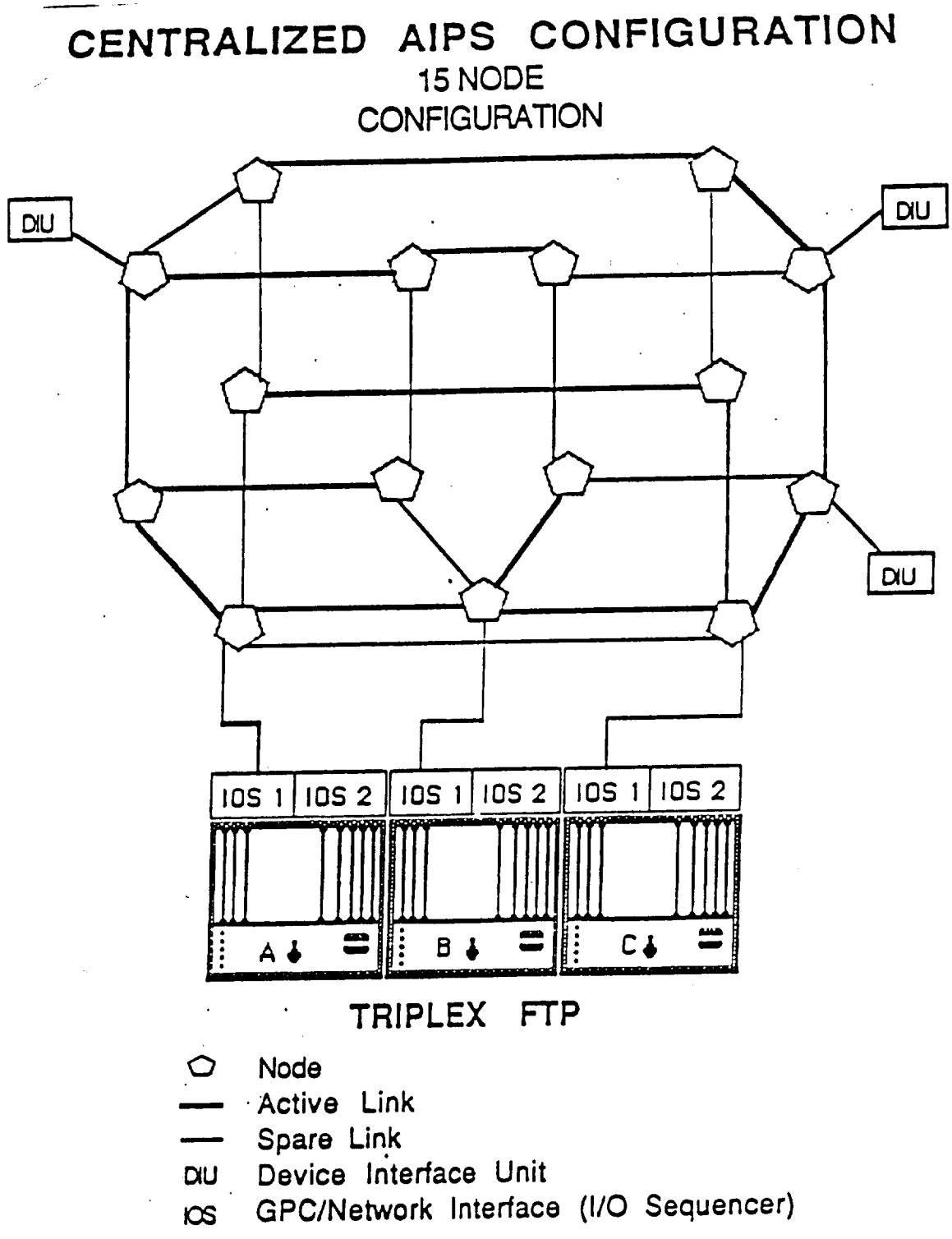


Figure 1.

Advanced Information Processing System IC Network

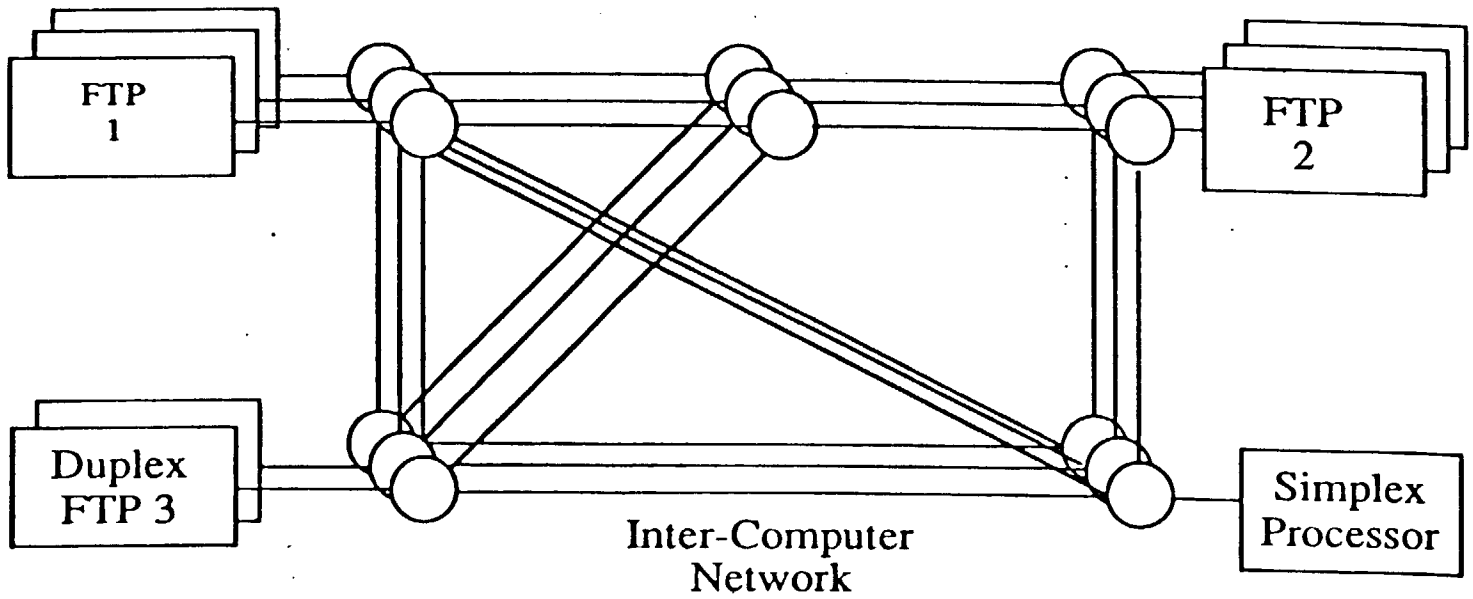
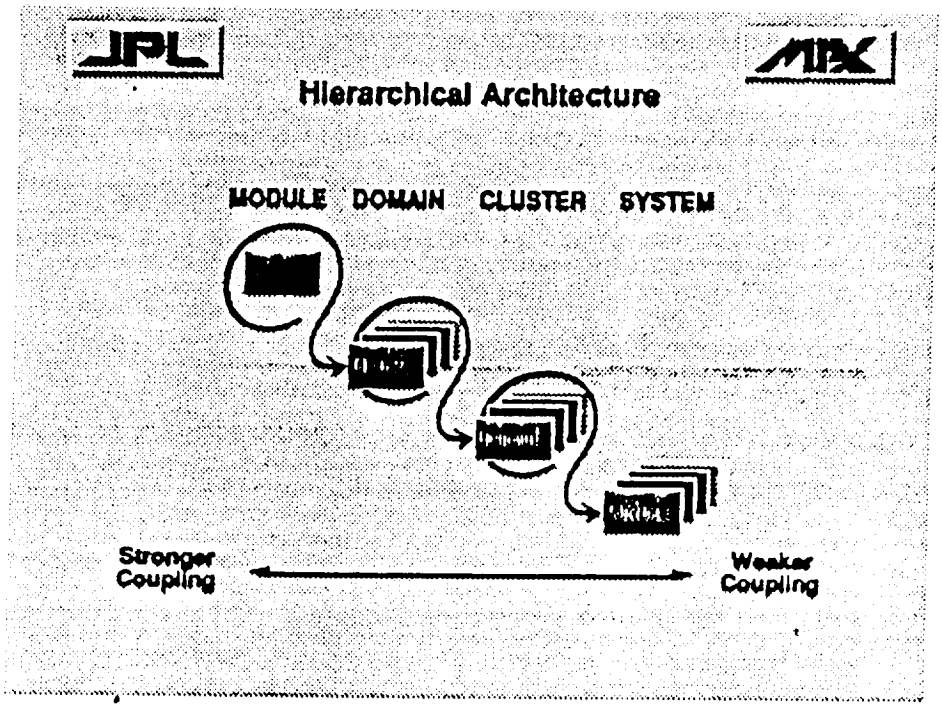


Figure 2.

MAX Fault Tolerance



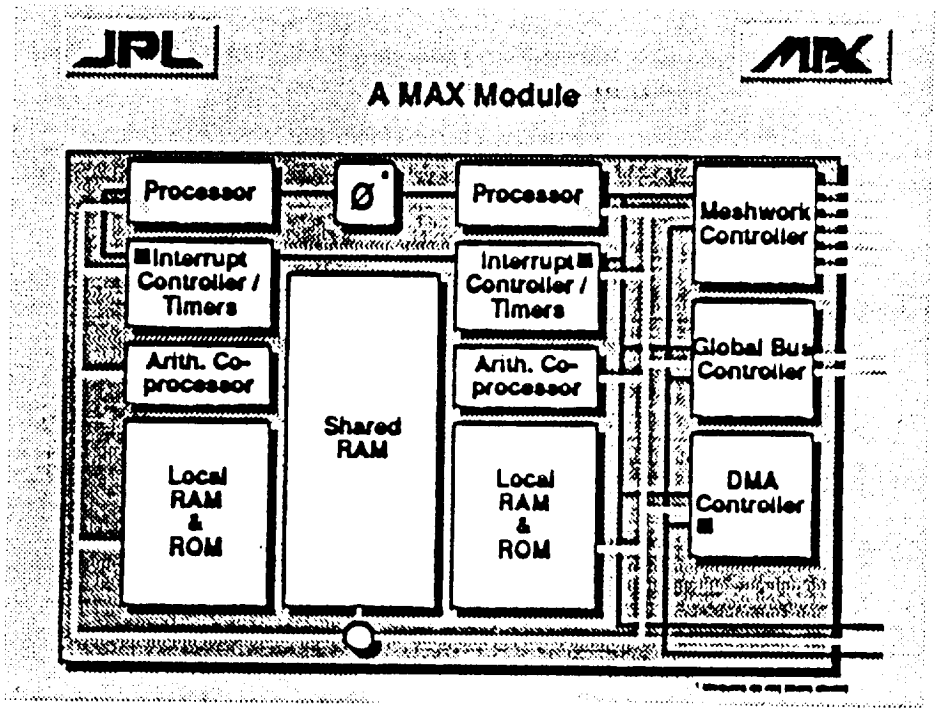
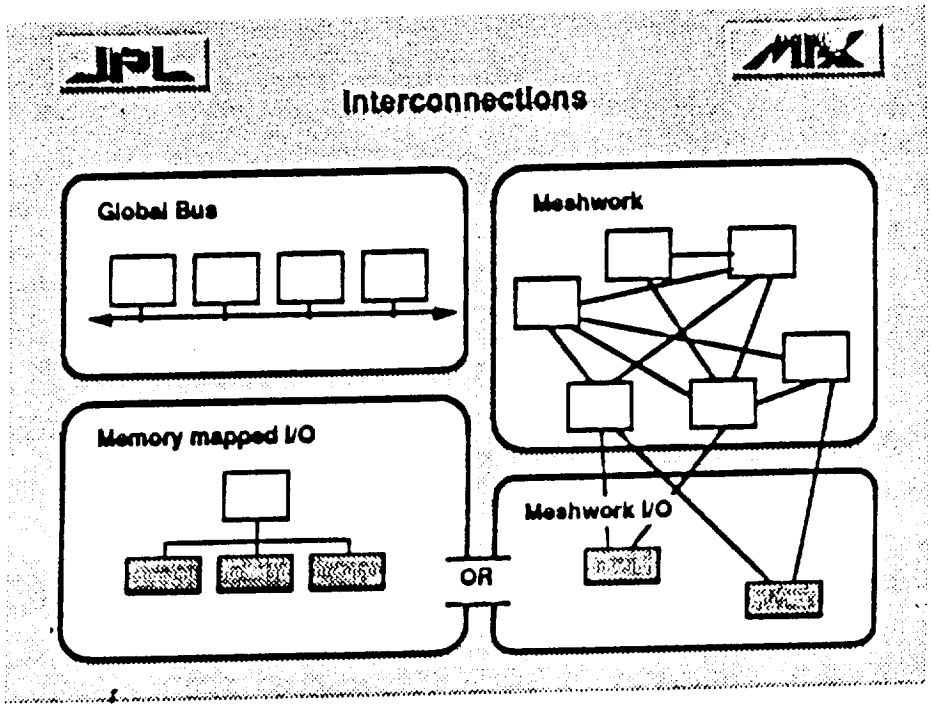


Figure 3.

MAX Fault Tolerance



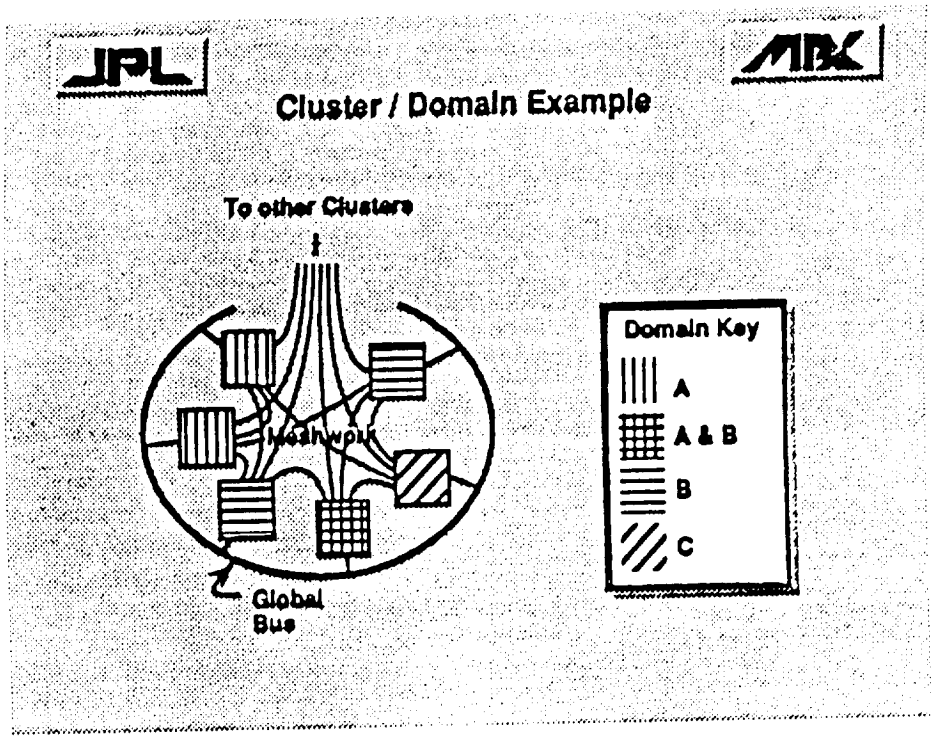
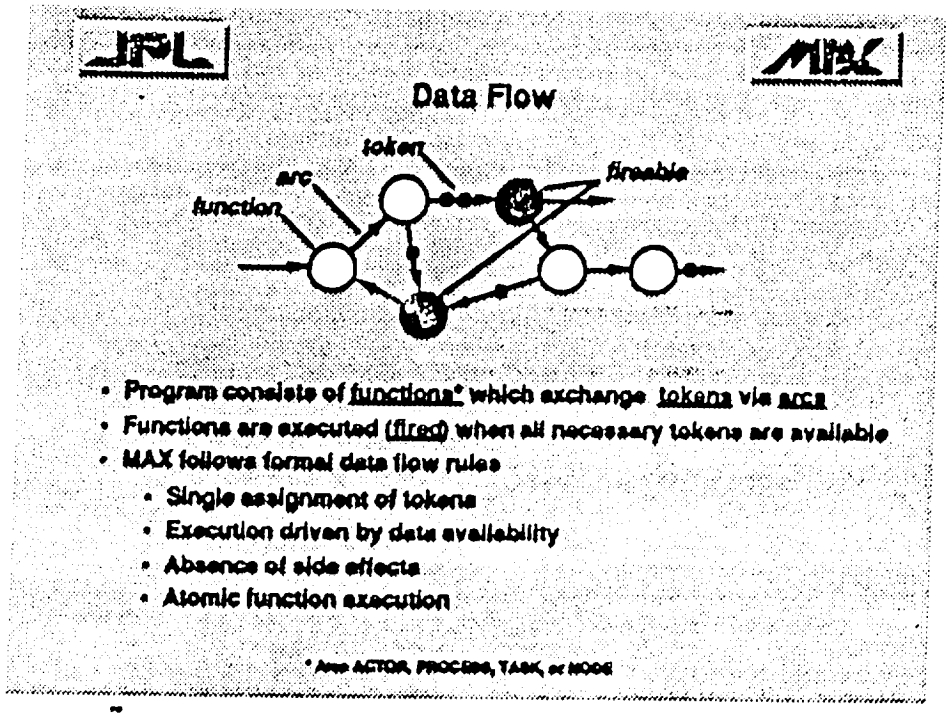


Figure 4.

MAX Fault Tolerance



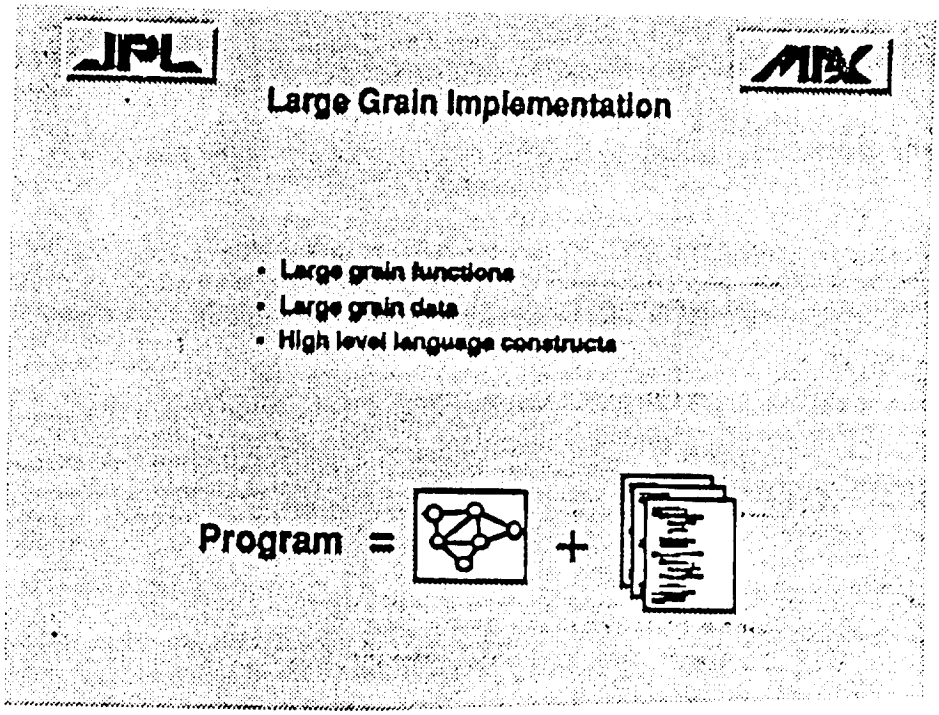


Figure 5.

Appendix I: Task Assignments

1. Task Assignments for the Graduate Assistants

Name: Myung-Hee Kim **Period:** Jan. 15 thru May 31, 1989

Mrs. Kim, supervised by Dr. Cezzar, assisted in research aspects by collecting articles and brochures in parallel processing/artificial intelligence areas mentioned in the proposal. She used Hampton University's Huntington Library and its CDROM facilities, the libraries of ODU and William and Mary, and NASA LaRC Technical Library. As needed, she wrote letters and make phone calls to collect the information. She had, in addition assisted Dr. Cezzar in developing a simulator (written in C or Pascal) for testing various process management strategies in shared-memory and message-passing (e.g. hypercube) machines. She used the opportunity to develop skills in programming and gain knowledge in computer sciences.

Name: Seema Farhat **Period:** Jan 15 thru Feb 30, 1989

Mrs. Farhat, supervised by Dr. Sen, assisted in research aspects by handling periodicals and subscriptions as needed, and other material such as textbooks and conference proceedings relevant to research areas mentioned in the proposal. In addition, she assisted Dr. Sen in implementing graph-mapping/optimization software on IBM PC's in the microprocessor lab. She used the opportunity to develop skills in programming and gain knowledge in computer sciences.

Name: Nelson Mygina **Period:** May 1 thru August 30, 1989

Nelson D. Mygina's main responsibility was to develop a simulator program for testing various process management policies in parallel operating systems. He used VMS Pascal to accomplish the task on the computer accounts provided in Hampton University Academic Computer Center VAX 8350 VMS machine.

2. Task Assignments for the Undergraduate Assistants

Name: Sherman White, Jr. **Period:** May 1 thru July 31, 1989

We have hired Sherman White as an undergraduate assistant to replace Nelson Mgcina who left the project and returned to his native country. Sherman will conduct the multiprocessor performance simulator under Dr. Cezzar's supervision. His work will contribute to the PAPER project's Phase 3 in determining efficiency of system software for parallel machines.

Name: Marletta Snowden **Period:** June 1 thru July 31, 1989

As of June 1, 1989, Mrs. Myung-Hee Kim has taken a research position at the

Mathematics Department. We have hired Marletta Snowden to replace her and work throughout the Summer on the PAPER project. Marletta will be supervised by Dr. Sen in carrying out tasks such as developing the LISP-based simulator for applicative caching and graph embedding.

3. Task Assignments for the Research Faculty

Ranjan K. Sen, Co-principal Investigator

1) Study of AIPS Systems' Intercomputer Communication User Services.

The problem of reconfigurability in the presence of faults in the system will be studied. The AIPS model (15-node, each node made up of 3-triplex and 1-simplex FTMPS) and Hypercube systems will be the focus.

2) Plans for Distributed Redundancy Management for FY 89

The main idea is to get a feedback from Rich Harper, CSDL regarding distributed redundancy management for AIPS system. Most of this would be covered during our planned visit.

3) Programming Paradigms for Applicative Caching on Multiprocesors

This work is related to the development of programming for applicative caching on dataflow machines. Presently, the simulator in LISP of the Amamiya's dataflow system will be installed. Later, the problem will be studied in multiprocessor systems. The basis of such investigation lies in the effectivity of applicative caching in reducing computational cost when recomputations of functions due to reccursive referencing are involved.

Robert A. Willis, Jr., Advisory/Consulting Faculty

1) Software Implemented Fault Tolerance (SIFT)

The reliability and verifiability of software used for recovery and reconfiguration of fault-tolerant systems. Usually, this involves the operating system used for systems that have hardware fault-tolerance built-in.

2) Software Fault Tolerance Through Redundant Modules

These usually follow hardware concepts for software and use design diversity or N-version software modules

3) Self-Testing and Repairing Computer -Software Issues

This is another concept in reliability (really availability) of systems using both hardware fault tolerance and software methods

Ruknet Cezzar, Principal Investigator

1) Extract relevant parts of AIPS requirements study for unmanned planetary and deep-space probes.

   This will involve cross-checking about technology survey and the suggested AIPS architecture in 1984. An analysis should be made about AIPS trends then (1984) and now (1989).

2) Identify JPL experimental architectures that can be used for unmanned planetary explorations involving fly-bys, landers, or rovers.

   This will involve looking into possible JPL architectures such as JPL-Star, JPL/Caltech Mark III Hypercube with 32-nodes, etc...

3) As a focus area for detailed work on a related topic (Phase 1 or Phase 2).

   This will involve specific papers in neural computing and a thorough review of what neural computing is all about.

Appendix II: Neural Computing Bibliography

Articles from Books and Proceedings:

[1] Grossberg, S., "Neural Networks And Natural Intelligence", MIT Press, May 1988, 544 pp.

[2] Anderson, J. A., and Rosenfeld, E., "Neurocomputing", MIT Press, 1988, 800 pp.

[3] Vemuri, V., "Artificial Neural Networks: TheoreticalConcepts", IEEE computer Society Press, July 1988, 160 pp.

[4] Caudill, M., and Butler, C., "Proceedings of the First International Conference on Neural Networks", IEEE Service center, Piscataway, N.J., June 1987, 300 pp.

[5] Widrow, B., and Baxter, R.A., "Learning Phenomena inLayered Neural Networks", IEEE-ICNN, Vol. II

[6] Hinton, G., McClelland, J., and Goodhill, G., "Learning Representationa by Recirculation", IEEE Service Center, Nov. 1987

[7] Anderson, J., "Neurocomputing", MIT Press, 1988

[8] Babcock, K.L., and Westervelt, R.M., "Stability andDynamics of Simple Electronic Neural Networks with Added Inertia" ,Physica 23D , 1986, 464-469, Notyh-Holland, Amsterdam

[9] Babcock, K.L., and Westervelt, R.M., "Complex Dynamics in Simple Neural Circuits", Harvard University

[10] Baldi, P., "Neural Networks, Orientations of the Hypercube and Algebraic Threshold Functions", IEEE Transactions on Information Theory, Dept. of Mathematics, University of California, San Diego, La Jolla, CA 92093

[11] Burr, D.J., "A Neural Network Digit Recognizer", Bell Comm. Res. ,1987

[12] Castelaz, P., Angus, J., and Mahoney, J., "Application of Neural Networks to Expert System and Command and Control Systems", Hughes Aircraft, Fullerton, CA

[13] Crawford, W., Myers, M., and Kuczewski, R., "Application of New Artificial Neural System Information Processing Principles to Patern classification", TRW, Rancho Carmel, San Diego, CA

[14] Cruz, C., Hanson, W.A., and Tam, J.Y., "Neural Network Emulation Hardware Design Considerations, Proceedings of 1987 IEEE First Annual International Conference on Neural Networks IBM Palo Alto Scientific Center, Palo Alto, CA 94304

[15] Denker, J., "Neural Network Models of Learning and Adaptation, AT&T Bell, Laboratories, Holmdel, NJ 07733, Phycia D, Vol. 22D, 1986, pp. 216-232

[16] Egecioglu, O., Smith, T.R., and Moody, J., "Computable Functions and Complexity in Neural Networks", Computer Science Dept., University of California, Santa Barbara, CA

[17] Feldman, J.A., "Neural Representation of Conceptual Knowledge", TR 189, June 1986, University of Rochester

[18] Graf, H.P., Jackel, L.D., Howard, R.E., Straughn, B., Denker, J.S., Hubbard, W., Tennat, D.M., and Schwartz, D., "VLSI Implementstion of a Neural Network Memory with Several Hundreds of Neurons", AT&T Bell Laboratories, Holmdel, NJ 07733

[19] Grossberg, S., and Gutowski, W.E., "Neural Dynamics of Decision Making Under Risk: Affective Balance and Cognitive-Emotional Interactions Source", Psychological Review, in press, 1986

[20] Grossberg, S., "Cooperative Self-Organization of Multiple Neural Systems During Adaptive Systems", BostonUniversity, Boston, MA 02215

[21] Hammerstrom,D., Bailey, J., and Rudnick, M., "Interconnect Architectures for WSI Neurocomputers", Oregon Graduate Research Center, 1987

[22] Hecht-Nielsoen, R., "Performance Limits of Optical,Electro-Optical, and Electronic Neurocomputers", HNC 5893 Oberlin Dr., San Diego, CA 92121, SPIE Optical and Hybrid Computing, Vol. 634, 1986

[23] Hirsch, M.W., "Convergence in Neural Nets", Dept. ofMathematics, University of California, Berkeley, CA 94720

[24] Hoffmann, G., "A Neural Network Model Based on the Analogy with the Immune Systems", Dept. of Physics and Microbiology, University of British Columbia, Vancouver, B.C.,Canada V6T 2A6

[25] Hubbard, W., et. al. "Electronic Neural Networks", AT&T Bell Laboratories, Holmdel, NJ 07733

[26] Hutchinson, J., and Koch, C., "Simple Analog and Hybrid Networks for Surface Interpolation: Neural Nrtworks for Computing" ed. T.S. Denber, pp. 235-239, AmericanInstitute of Physics, New York, 1986

[27] Kawamoto, A., and Anderson, J.A., "A Neural Network Model of Multistable Perception", Acta Phychologica 59 (1985), 35-65, North-Holland

[28] Keeler, J.D., "Comparison Between Sparsely Distributed Memory and Hopfield-Type Neural Network Models", summited to J. Cog Sci also RIAES Tech. Report 86.31

[29] Keeler, J.D., "Information Capacity of Hebbian Neural Networks", summited to Phys. Rev. Letters PACS Numbers:87.30, 89.70

[30] Lapedes, A., and Farber, R., "A Self-Optimizing,Nonsymmetrical Neural Net for Content Addressable Memory and Pattern Recognition", Physica 22D (1986), pp. 247-259

[31] Levine, D.S., "Neural Network Model of Temporal Order Effects in Classical Conditioning Modelling of Biomedical Systems", Elsevier Science Pub., 1986

[32] Levine, D.S., "A Neural Network Theory of Frontal Lobe Function", Proc. of the Eighth Annual Conf. of the Cognitive Science Soc., (Amherst, MA, 1986), Erlbaum

[33] Linsker, R., "From Basic Network Principles to Neural Architecture: Emergence of Spatial-Opponent Cells", Proc. Natl. Acad. Sci. USA Vol. 83, pp 7508-7512, October 1986

[34] Myers, M.H., "Some Speculations on Artificial NeuralSystem Technology", NAECON 1986 Proc.

[35] Omohundro, S.M., "Efficient Algorithms with Neural Network Behavior", Dept. of Comp. Sci., University of Illinois at Urbana-Champaign

[36] Pellionisz, A.J., "Sensorimotor Operations: A Ground for the Co-Evolution of Brain Theory with Neurobotics and Neurocomputers", Proc. IEEE 1st Ann. Intl. Conf. on Neural Networks, San Diego, CA, 1987 June

[37] Penz, P.A., "The Closeness Code: An Integer to Binary Vector Transformation Suitable for Neural Network Algorithms", Texas Instruments Inc., Dallas, TX Tomaso Poggio

[38] Reeke, G.N. Jr., and Edelman, G.M., "Selective Neural Networks and Their Implications for Recognition Automata", The Rockerfeller University

[39] Sage, J.P., Thompson, J., and Wither, R.S., "An Artificial Neural Network Integrated Circuit Based on MNOS/CCD Principles", MIT Lincoln Lab.

[40] Tank, D.W., and Hopfield J.J., "Neural Computation by Concentrating Information in Time", AT&T Bell Labs, 1987

[41] Venkatesh, S.S., and Psaltis, D., "Linear and Logarithmic Capacities in Associative Neural Networks", Preprint Sumitted March 1985 to IEEE Transactions on Information Theory, Revised Nov. 1986

[42] Voevodsky, J., "A Neural-Based Knowledge Processor",Neuraltech, Mountain View, CA

[43] Zipser, D., "Programming Neural Nets to Do SpatialComputations", ICS Report 8608, UCSD

[44] Anderson, J.A., and Rosenfeld, E., "Neurocomputing, A Reader", MIT Press, Cambridge, Mass., 1988

[45] Fukushima, K., Miyake, S., and Ito, T., "Neocognition: A Neural Network Model for a Mechanism of Visual Pattern Recognition", IEEE Trans. Syst. Man cybern. SMC-13,5(Sept.-Oct. 1983), pp. 826-834

[46] Hopfield, J.J., "Neural Networks and Physical Systems with Emergent Collective Computational Abilities", Proc. of the National Academy of Sciences USA. National Academy of Sciences, Washington, D.C., 1982, Vol. 79, pp. 2554-2558

[47] Widrow, B., "Generalization and Information Storage in Networks of Adaline 'Neurons'", in Self-Organizing Systems 1962, Yovitz, M.C., Jacobi, G.T., and Goldstein, G., eds., Spartan Books, Washington, D, 1962, pp. 435-461

[48] Sivilotti, M.A., Emerling, M.R., and Mead, C.A., "VLSI Architectures for Implementation of Neural Networks", Proc. Conf. Neural Networks for Computing, Denker, J.S., ed., 1986, American Institute of Physics Conf. Proc. 151, pp. 408-413

[49] Graf, H.P., and deVegvar, P.G.N., "A CMOS Implementation of a Neural Network Model", Advanced Research in VLSI, Proc. Stanford Conf. 1987, Losleben, P., ed., MIT Press, Cambridge, Mass., 1987, pp. 351-367

[50] Sage, J.P., Thompson, K., and Withers, R.S., "An Artificial Neural Network Integrated Circuit Based on MNOS/CCD Principles", Proc. Conf. Neural Networks for Computing, 1986, Denker, J.S., ed., American Institute of Physics Conf. Proc. 151, pp. 381-385

[51] Hopfield, J.J., "Neurons with Graded Response HaveCollective Computational Properties Like Those of Two State Neurons ",Proc. Academy of Science USA, Vol. 81, 1984, pp.3088-3092

[52] Tank, D.W., and Hopfield, J.J., "Simple 'Neural'Optimization Networks: An A/D Converter, Signal Decision Circuit, and Linear Programming Circuit", IEEE Tran.Circuits and Systems, Vol. CAS-33, 1986, pp. 533-541

[53] Lippmann, R.P., "An Introduction to Computing with Neural Nets", IEEE Acoustics, Speech, and Signal Processing, Vol. 4, April 1987, pp. 4-22

[54] deVegvar, P.G.N., and Graf, H.P., "Studies of Associative Memory and Sequence Analyzer Circuits on a Programmable Neural Network chip" to be published. Contact Graf for preprint copies.

[55] Linsker, R., "From Basic Network Principles to Neural Architecture", Proc. Natl. Academy of Sciences USA, Vol. 83, Oct.-Nov. 1986, pp. 7508-7512, 8390-8394, 8779-8783

[56] Linsker, R., "Towards an Organizing Principle for a Layered Perceptual Network", in Anderson, D., ed., "Neural Information Processing Systems-Natural and Synthetic", Amer. Inst. of Physics (NY), to appear

[57] Willows, A.O.D., Dorsett, D.A., and Hoyle, G.,"Neurobiol", 1973, 4, 207-237, 255-285

[58] Harmon, L.D., "Neural Theory and Modeling", Reiss, R.F., ed., Stanford University Press, Stanford, CA, pp. 23-24

[59] Cowan, J.D., and Sharp, D.H., "Neural Nets and Artificial Intelligence", Daedauls 1988; 117:85-122

[60] Castelaz, P.F., "Application of Neural Network Solution to Battle Management Processing Functions", 55th Military Operations Research Society Symposium, Huntsville AL, May 1987

[61] Denker, J.S., ed., "Neural Networks for Computing",American Institute for Physics (NY), 1986

[62] Brown, N.H. Jr., "Neural Network Implementation Approaches for the Connection Machine", Conf. on Neural Information Proc. Systems-Natural and Synthetic, 1987

[63] Hastings, H.M., and Waner, S., " 'Neural Nets on the MPP', Frontiers of Massively Parallel Scientific Computation", NASA Conference Publication 2478, NASA Goddard Space Flight Center, Greenbelt Maryland, 1986

Articles from Technical Journals:

[1] Kohonen, T., "Adaptive, Associative, and Self-Organizing Functions in Neural Computing", Applied Optics 26(23): Dec. 1, 1987

[2] Baldi, P., and Venkatesh, S.S., "Number of Stable Points for Spin-Glasses and Neural Networks of Higher Orders", Physical Review Letters Vol. 58, Number 9, 1 March 1987

[3] Fukushima, K., and Ito, T., "A Neural Network Model Extracting Features from Speech Signals", The Transactions of the Institute of information and Communication Engineers, Japan, Vol. J70-D, No. 2, pp. 451-462

[4] Golden, R.M., "A Developmental Neural Model of Visual Word Perception", Cognitive Science 10, 1986, pp. 241-276

[5] Golden, R.M., "The 'Brain-State-in-a-Box' Neural Model Is a Gradient Descent Algorithm", Journal of Mathematical Psychology Vol. 30, No. 1, March 1986

[6] Grossberg, S., and Mingolla, E., "Computer Simulation of Neural Networks for Perceptual Psychology", Behavior Research Methods, Instruments, and Computers 1986, 18(6), pp. 601-6-7

[7] Sasiela, R., "Forgetting as a Way to Improve Neural-Net Behavior", American Inst. of Physics, Neural Networks for Computing 1986

[8] Shaw, G.L., and Roney, K.J., "Analytic Solution of a Neural Network Theory Based on an Ising Spin System Analogy", Physics Letters, Vol. 74A, Number 1,2, 29 Oct. 1979

[9] Sompolinsky, H., "Neural Networks with Nonlinear Synapses and a Static Noise", Physical Review, Vol. 34, No. 3, Sept. 1986

[10] Abu-Mostafa, Y.S., and Psaltis,D., "Optical Neural Computers", Scientific American, March 1987, pp. 88-95

[11] Fukushima, K., "A Neural Network Model for Selective Attention in Visual Pattern Recognition", BiologicalCybernetics, Vol. 55, No. 1, Oct. 1986, pp. 5-15

[12] Fukushima, K., "A Neural Network Model for Selective Attention in Visual Pattern Recognition and Associative Recall", Applied Optics, Vol. 26, No. 23, Nov. 1987, pp. 4985-4992

[13] Fukushima, K., "Cognitron: A Self-Organizing Multilayered Neural Network", Biological Cybernetics, Vol. 20, No. 3/4, Nov. 1975, pp. 121-136

[14] Fukushima, K., "Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position", Biological Cybernetics, Vol. 36, No. 4, April 1980, pp. 193-202

[15] Bruce, C., Desimone, R., and Gross, C.G., "Visual Properties of Neurons in a Polysentory Area in the Superior Temporal Sulcus of the Macaque", J. Neurophysiology, Vol. 46, No.2, Aug. 1981, pp. 369-384

[16] Carpenter, G.A., and Grossberg, S., "A Massively Parallel Architecture for a Self-Organizing Neural Pattern Recognition Machine", Computer Vision, Graphics and Image Processing, Vol. 37, No. 1, Jan. 1987, pp. 54-115

[17] Fukushima, K., "Neocognitron: A Hierarthical Neural Network Capable of Visual Pattern Recognition", Neural Networks, Vol. 1, No. 2, April 1988

[18] Hopfield, J.J., and Tank, D.W., "Computing with Neural Circuits: A Model", Science, Aug. 8, 1986

[19] Feldman, J.A., "Dynamic Connections in Neural Networks", Biological Cybernetics, Vol. 46, 1982, pp. 27-39

[20] Oja, E., "A Simplified Neuron Model as a Principal Component Analyzer", J. Math. Biology, Vol. 15, 1982, pp. 267-273

[21] Woods, W. A., "Transition Network Grammars for Natural Language Analysis", Comm. ACM, Vol. 13, No. 10, Oct. 1970, pp. 591-606