RECONFIGURATION FOR FAULT TOLERANCE AND PERFORMANCE ANALYSIS

Harold Henry Kollmeier

MS-CIS-87-106

Department of Computer and Information Science
School of Engineering and Applied Science
University of Pennsylvania
Philadelphia, PA 19104-6389

November 1987

Acknowledgements: This research was supported in part by DARPA grants NOOO14-85-K-0018, NSF grant MCS-8219196-CER and U.S. Army grants DAA29-84-K-0061, DAA29-84-9-0027.

UNIVERSITY OF PENNSYLVANIA
THE MOORE SCHOOL OF ELECTRICAL ENGINEERING
SCHOOL OF ENGINEERING AND APPLIED SCIENCE

RECONFIGURATION FOR FAULT TOLERANCE AND PERFORMANCE ENHANCEMENT : A COMPARATIVE ANALYSIS

Harold Henry Kollmeier

Philadelphia, Pennsylvania
May, 1987

A thesis presented to the Faculty of Engineering and Applied Science of the University of Pennsylvania in partial fulfillment of the requirements for the degree of Master of Science in Engineering for graduate work in Computer and Information Science.

Yuen-wah Eva Ma

O. Peter Buneman

Reconfiguration for Fault Tolerance and for Performance Enhancement: A Comparative Analysis

Abstract

Architecture reconfiguration, the ability of a system to alter the active interconnection among modules, has a history of different purposes and strategies. Its purposes develop from the relatively simple desire to formalize procedures that all processes have in common to reconfiguration for the improvement of fault-tolerance, to reconfiguration for performance enhancement, either through the simple maximizing of system use or by sophisticated notions of wedding topology to the specific needs of a given process. Strategies range from straightforward redundancy by means of an identical backup system to intricate structures employing multistage interconnection networks.

The present discussion surveys the more important contributions to developments in reconfigurable architecture. The strategy here is in a sense to approach the field from an historical perspective, with the goal of developing a more coherent theory of reconfiguration. First, the Turing and von Neumann machines are discussed from the perspective of system reconfiguration, and it is seen that this early important theoretical work contains little that anticipates reconfiguration. Then some early developments in reconfiguration are analyzed, including the work of Estrin and associates on the "fixed plus variable" restructurable computer system, the attempt to theorize about configurable computers by Miller and Cocke, and the work of Reddi and Feustel on their restructable computer system.

The discussion then focuses on the most sustained systems for fault tolerance and performance enhancement that have been proposed. An attempt will be made to define fault tolerance and to investigate some of the strategies used to achieve it. By investigating four different systems, the Tandem computer, the C.vmp system, the Extra Stage Cube, and the Gamma network, the move from dynamic redundancy to reconfiguration is observed. Then reconfiguration for performance enhancement is discussed. A survey of some proposals is attempted, then the discussion focuses on the most sustained systems that have been proposed: PASM, the DC architecture, the Star local network, and the NYU Ultracomputer. The discussion is organized around a comparison of control, scheduling, communication, and network topology.

Finally, comparisons are drawn between fault tolerance and performance enhancement, in order to clarify the notion of reconfiguration and to reveal the common ground of fault tolerance and performance enhancement as well as the areas in which they diverge. An attempt is made in the conclusion to derive from this survey and analysis some observations on the nature of reconfiguration, as well as some remarks on necessary further areas of research.

Table of Contents

Figures

1. INTRODUCTION

Among the well known issues in computer design is system reconfiguration, but in spite of being well known it has developed little focus, remaining instead at a level of proliferation of different purposes and design strategies. There are, of course, some aspects of reconfiguration about which there is agreement. It has been defined as a condition under which a system may assume several architectural configurations, each of which is characterized by its own topology of activated interconnections between modules [Sie79b]. And it can be agreed that reconfiguration by its very nature makes subsystems out of larger systems, for different purposes, traditionally for fault tolerance and more recently for performance enhancement.

Some aspects of reconfiguration, of course, remain without agreement. Perhaps the greatest indication of the state of thinking about reconfiguration is the traditional understanding that reconfiguration means many things, and that it is usually an adjunct to other concerns. As a design problem it certainly does not exist alone, and discussions of reconfiguration will very often be found in the literature on SIMD and MIMD research, partitionable architectures, and parallel processing. It has been said to be a state change that is effected without human intervention [Ma82], although work has been done to allow control "explicitly," by the high-level programmer [Sch86]. The very proliferation of proposals for widely different architectures all coming under the umbrella of a similar purpose suggests the variety of perspective. And while the term *reconfigurable* is widely understood, it is not in universal use in discussions of this design issue: other possibilities include *dynamic architecture* ([Kar86b]), *restructurable* [Red78], and *configurable* [Sny82].

For our purposes in this investigation, the two different, major purposes for reconfiguration - fault tolerance and performance enhancement - provide the most interesting focus for investigation. Fault tolerance, the ability of a system to continue operation under less than maximum and perhaps increasingly degrading conditions, and performance enhancement, the attempt to match systems to advanced processing demands,

have separately developed strategies for reconfiguration. But their similarities and the space where they come together that is the focus of this investigation.

Many early developments in computer technology display a primitive version of what might be called reconfiguration, in that they formalized system alteration that occurred as a result of I/O control, secondary storage access, overlaying, and other procedures that processes have in common. A system that has more software and hardware components than are needed for a specific task must therefore be configured for that task; that is, the subset of the system that is needed for the task must be created. As systems became more complicated and time-sharing became standard, the forming of subsets of the overall entity became part of the formal thinking on software control. The early PDP-11 handbook, for example, in its discussion of the innovative abilities of the UNIBUS to allow bidirectional and asychronous communication between any two connected modules, perceives of the machines capabilities as a form of reconfiguration [Dec76]. But we must bear in mind here that this is only a simple, primitive version of what we are calling "reconfiguration," and that more sophisticated strategies follow.

When Denning presents the theory of virtual memory as a disassociating of physical address space and logical address space, he is speaking of the reconfiguring of the system into subsets [Den70]. Fault-tolerance is the next step in this development, whereby subsets of the system form redundant parts allowing for continued operation when components fail. Fault-tolerance is still very much at the forefront of thinking on reconfiguration (e.g., [Sie82]), but added to this are concerns over the use of reconfiguration for performance enhancement, either through the simple maximizing of use of the entire system, or by the more sophisticated notion of wedding topology to the specific needs of a given process: this has been referred to as enhancing the degree of "match" between algorithm and architecture [Yal85]. System reconfiguration - the creating of subsets that will be more in tune with a specific task than is the entire system - stands in opposition to the trend toward dedicated systems.

The concern of the present study is system reconfiguration for the sake of performance enhancement as well as fault tolerance, with an emphasis on multiprocessor environments. The issues involved in system reconfiguration are many. Control is a dominant concern, for the creating of subsets within a system brings up the problem of individual unit performance in coordination with the whole. A particular aspect of control is scheduling, for maximum use of the system but also for problems of synchronization when the purpose of the system is parallel processing. Communication needs are strong when reconfiguration occurs in a multiprocessing environment, and much of the literature concerns itself with the interconnection networks that are necessary in a reconfigurable system. Another major issue is precisely when and where the reconfiguration will occur; among the more interesting developments here is the research into revising the traditional high-level languages to support programmer- controlled configuration [Kuc85] [Cli85] [Arv80] [Ree80]. Designs for reconfiguration are also controlled, or it seems they should be, by the purpose for which the system is being developed. Many proposals, some more developed than others, responding to these issues and to the need for reconfigurable systems, have appeared in the literature.

Our strategy here is to approach the field from an historical perspective, with the goal of developing a better understanding of reconfiguration. First, the Turing and von Neumann machines will be discussed from the perspective of system reconfiguration, and it will be seen that this early and important theoretical work contains little that anticipates reconfiguration. One intention in this analysis is to develop the theme that reconfiguration, unlike other major developments in the technology, proceeds without a theoretical base. We will focus on some key developments in reconfiguration, which include the work of Estrin and associates on the "fixed plus variable" restructurable computer system. We will then discuss an interesting attempt by Miller and Cocke to theorize about configurable computers. We also review the work of Reddi and Feustel on their restructable computer system. This section of the paper is therefore not so much a survey as a close look at some key developments. The discussion will then focus on the most sustained systems for fault

tolerance that have been proposed. An attempt will be made to define fault tolerance and to investigate some of the strategies used to achieve it. We will see that a distinction can be made between the early strategies leading up to what Siewiorek calls "dynamic redundancy" [Sie82] and the later developments that make use of strategies beyond those in Siewiorek's scheme, including systems that employ multistage interconnection networks. By investigating four different systems, the Tandem computer, the C.vmp system, the Extra Stage Cube, and the Gamma network, we will see the move from dynamic redundancy to the more advanced version of reconfiguration that is our interest here. Indeed, it would be appropriate to invent new terminology to describe the more sophisticated strategies that we will be discussing.

Discussion of performance enhancement and its relation to reconfiguration will then be attempted, through a survey and analysis of some of the more significant proposals. Some of these design have reached fruition in the form of working machines, if only in prototype; others remain paperwork machines, which, however, contribute in their own way to the development of thinking on the subject. Interest in these new designs results from the realization that the architecture concepts and technology of the now fully developed high performance "von Neumann" machines will not match the demands for massive processing that are present in such fields as image processing and supercomputing. The issue of reconfiguration for performance enhancement aligns itself strongly with issues of parallel processing, including the issue of interconnection networks. A survey of some of these many proposals will first be attempted, in order to give the reader a sense of the range of ideas on the subject, and in order to provide a contrast to the proposals for reconfiguration for fault tolerance. Then the most sustained systems that have been proposed will be discussed under the two issues of communication and control. These developments include the dynamic architecture of the Kartashevs, PASM, the Star local network, and the NYU Ultracomputer.

Reconfiguration for performance enhancement is perhaps a stronger concern in this study than is reconfiguration for fault tolerance, but the comparison of the two issues

should reveal their common ground as well as the areas in which they diverge. An attempt will therefore be made in the conclusion to derive from this survey and analysis some observations on the nature of reconfiguration, as well as some remarks on necessary further areas of research. It is hoped that these effort will provide the groundwork for a more accurate understanding of the topic.

2. RECONFIGURATION IN THE EARLY LITERATURE

2.1 Strategy of the Present Section. The discussion here begins our historical analysis of reconfiguration. By looking at some early work, both in theory and in the development of design proposals, we will be able to formulate some fundamental premises upon which to proceed with the analysis of later developments. We will see that many of the motivations for reconfiguration appear early in the literature, but that computer applications had not yet sufficiently developed, particularly in areas of image processing and related matters in robotics, to allow a fully developed set of motivations and criteria. We will also see that reconfiguration appears very little in the early thinking on computing, because aspects of finite time and finite space are not relevant to that thinking. Reconfiguration rises late, relatively speaking, in the development of the technology; it rises as a response to problems in the technology itself, rather than as a response to the very nature of algorithms and problem solving.

In order to proceed with these observations we will look at three different sections of early developments. First we will examine, with an eye on reconfiguration, the early classic thinking on computation, the well known presentations of Turing and von Neumann. The question here is, in this early, famous theorizing on the nature of computation, is there anything that anticipates reconfiguration? Next we will look at an article from the mid 70s by Miller and Cocke, which attempts to provide a theoretical framework for developing notions of reconfiguration. Finally, we will analyze and compare two early proposed systems that of Estrin and associates and that of Reddi and Feustel.

2.2 Early Theory in Computation. In developing an understanding of reconfiguration, we would tend to look back to the early thinking on computing, but in doing so we will find that there is very little in the classic literature that suggests reconfiguration. We could take the work of Turing and of von Neumann as central here.

The Turing machine, as originally presented [Tur36], is the classic of sequential processing. State change is effected by the linear movement of the squares of a tape

through the "machine." The machine is able to read, or to scan, the square of the tape that it was at the moment focusing on, or that was "in" the machine. The symbol set of the tape was limited to 0, 1, and empty. The machine could read and write symbols, but it could also erase them; and while it could only move from one square to the next, it could go backwards and forwards and it could move over a square without altering it, so that its domain was the infinite tape. The combinations of reading, writing, erasing, and scanning gave the machine a finite set of states, which Turing called its "*m*-configurations." Through this behavior the machine was capable of memory, in that it could move to a previously scanned and (perhaps) altered square, and thereby "recall" what was there. It could also perform arithmetic, through a process of copying and erasing.

The advances of the Turing machine over the more simple automata, including its left and right movement and its ability to mark squares, are ingenious, but they do not take the idea of processing beyond the sequential. Perhaps the most important reason that the Turing machine is not concerned with parallelism and reconfiguration is the fact that time and space are not issues in the machine: the tape that passes through the reading and writing head is potentially infinite, and computation, while always finite, can go on indefinitely in Turing's theoretical context.

The first presentation of the computer by von Neumann and associates [Bur46] is not a theoretical paper but rather an astonishingly complete description of the logical design of the sequential machine. Its importance, however, has propelled it into the realm of theory in the field. It is important as "theory" partly because it establishes time and space as important to the fundamental thinking about computing machines. The paper presents a practical core of considerations on memory storage, control and machine/human communication, and arithmetic. The change of state consists of the movement from one instruction to the next, under the control of the Control Register and the Control Counter. It establishes the notion of the machine as an instrument of strict sequential code execution with no distinction in the internal representation of different data types. The only hint of processing beyond the strictly sequential is the suggestion in the paper of a method of error

checking whereby two identical computers, controlled by the same clock, operate in parallel and check each other's results.

2.3 Miller and Cocke's Theory of Configurable Computers. At this point we will pause to consider not a proposed design, but rather the attempt by Miller and Cocke in the early 70s to present a theory of the "configurable" computer [Mil74]. The attempt is interesting because in addition to the principles it lays out, it also presents a class of configurable computers, called "search mode configurables," which do not make use of an interconnection network; this description shows a strategy that seems to have been lost in further thinking of reconfiguration, and reminds us that there was a time when reconfiguration was not necessarily wed to the problem of interconnection networks.

Miller and Cocke observe that all developments up until that time have not changed the fundamental von Neumann concept of the stored program: innovations have removed bottlenecks and improved performance, but the von Neumann machine remains. For Miller and Cocke, the most important implication of the stored program machine is that the program must be used to mold the algorithm to the fixed structure of the machine. That is, the program is used to sequence the program operation; or machine first, algorithm second. Miller and Cocke regard the new class of configurable computers to be a major departure from this traditional stored program approach, while still making use of notions of the program, high-level languages, compiler techniques, etc. The important motivation in configurable computers is that "the machine structure should attain the natural structure of the algorithm being performed." The advantage to these proposed machines is that they will enjoy the speed enhancement found in special purpose machines, but also not discard the advantages of general purpose machines. Configurable computers also enhance the development of parallel execution.

These and other advantages are found in two classes of configurable computers, the search mode configurables and the interconnection mode configurables. A search mode configurable, as pictured in Figure 1, is a multiprocessing system with three parts, a set of operational units, memory, and a searcher.

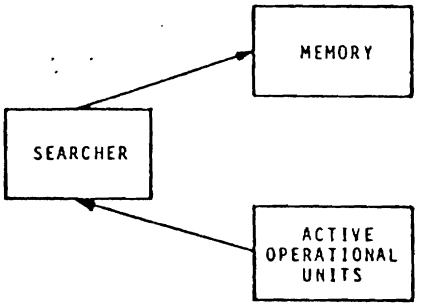

Figure 1 - The Search Mode Configurable Computer

If we are thinking with hindsight and therefore conceiving of an interconnection network as an inevitable part of a reconfigurable system, the searcher takes the place of the network. When an operational unit has finished a task it requests the searcher to find a suitable task in memory for it to process next. Tasks in memory are conceived of as data containing internal information, via an operation code and a tag, that identifies the data as an appropriate task for a given operational unit. The searcher therefore searches memory to find a unit of data that is a suitable match for the requesting operational unit. During processing, an operational unit may change the internal information stored with the data, thereby returning the unit to memory with information that destines the unit for further processing by another operational unit.

Clearly, in this multiprocessing environment parallel processing of different units of the same algorithm can take place. The bottleneck switches from processors to the searcher, but the design allows for a multi-searcher system. The searcher, in addition to performing as a processor, might seem to be an interconnection network, except that the kind of processing it performs and the presence of internal information in the data in memory effect memory/processor relationship.

The alternate possibility presented by Miller and Cocke, called by them the interconnection mode configurables, is closer to what we would normally understand to be a reconfigurable system structure. This is for the simple reason that the heart of the matter is the now well understood interconnection network, the ICN. See Figure 2.

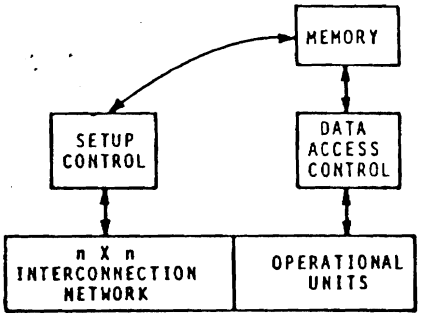

Figure 2 - The Interconnection Mode Configurable Architecture

Instead of having the searcher connect memory and the operational units, the operational units themselves are connected to one another, depending on appropriate interconnection based on analysis of the algorithm. This connecting is done by the interconnection network; the interconnection network can thus be seen as replacing the searcher, or can be seen as a refinement or further development of the searcher design. Access frequency to memory is therefore diminished, because completion of a process in a given unit does not here mean return of data to memory, as in searcher mode, but rather movement of data via the interconnection network to the next operational unit, in a manner that bears similarity to data-flow architecture.

The high-level language program is first compiled into blocks of a size suitable for use of the operational units. The compiler then works sequentially with these blocks. The compiler performs a type of data flow analysis on the given block, and then establishes a setup procedure for the block; the setup procedure is basically the flow of operation for the interconnection network. The setup procedure is stored in memory as an instruction, and is the first instruction of a block. All instructions have been accessed during the execution of a block, and therefore memory access need only occur for operands and results. Completion of the execution of a block means exit from the block and initializing of the setup of the next block. This scheme is therefore based on the notion of preanalysis and an establishing of all patterns of interconnection before run time. Each block can be conceived of as determining a special-purpose machine that exists for the duration of its

own execution. We will see that this model anticipates the later work of the Kartashevs on their dynamic architecture [Kar79a].

The theory of Miller and Cocke does not establish with any detail a complete system for reconfiguration; but it does show the fundamental workings that others will develop more fully. The search mode appears to be an early development, overtaken by the more valuable interconnection mode; and we will see that most later proposals are built on this model.

2.4 A Comparison of Two Early Designs. Some early work that deserves attention is that of Estrin and associates on the *F* plus *V* (fixed plus variable) machine [Est60] [Est63a] [Est63b]. There is no need in the present context to review the details of planned implementation in the design, although plans for scheduling and human interaction [Est63b], as well as the strategy of physical changing the wiring harness that connects modules to effect reconfiguration [Est63a] are of interest. What is important here are the motivations established by Estrin for the development of a reconfigurable system -- or rather, in his presentation, a "restructurable computer system" -- as well as some of the notions of how the design should proceed.

The issue for Estrin is practicable computability, and the problems that fall outside its domain [Est63a]. Practicable computability is a function, among other things, of cost, limit of size, time, and machine reliability. While advances up until the early sixties had increased the number of problems that could be called practicably computable, the number that was not was still large. Coupled with the inherent, finite limits of the machine was the demand placed on it to be general purpose.

Estrin saw this as a further restriction in an already limited environment. The general purpose computer is a compromise in establishing of word length, selection of arithmetic algorithms, and determination of instruction set. The desire to serve a wide variety of problems prevents the general purpose machine from developing into a system that has the speed or size necessary to solve the problems that remain outside the domain of the practicably computable.

The solution that had been developing at the time of Estrin's proposal was the building of the special purpose computer: general purpose problem solving was slighted in the favor of machines that were constructed for the fast and efficient solution of restricted classes of problems. Thus, the domain of the solvable was, according to Estrin, not restricted so much by available technology as by the demand of general purpose.

Paradoxically, therefore, the domain of the solvable could be expanded by limiting the number of problems that a given system could solve. Of course the drawback here was also evident: the special purpose computer does not respond readily to changes in problem formulation, solution methods, or computational needs. By establishing a system that does a few things well the numbers of things it does not do well increases, and the likeliness increases, given the range of problems that need to be solved, that the machine will enter a state in which it is not performing efficiently. There is also the practical problem of catering to an audience large enough to provide the means for development of an inevitably expensive system that provides only limited problem solving.

Estrin offered the following premises for the development of a new system that would address these matters:

> 1. In the solution of any given problem, a special purpose computer can be built to be more efficient than a general purpose computer.
> 2. The essential sequential form of many algorithms contains parts which may be executed simultaneously on different processors with a consequent reduction of the computation time.
> 3. Within the constraints of a finite hardware inventory, a greater number of computing substructures can be built if the inventory is restructurable than if it is committed to a nonvariable system.
> 4. Writing a compiler program for a large computer system is an effort measured in man years and is practical only if the computational characteristics (*e.g.*, instruction list and meaning of instructions) remain essentially fixed over the lifetime of the system [Est63a].

Estrin's response to his own premises was the proposed fixed plus variable computer [Est60]. Attempting to combine the advantages of both general purpose and special purpose schemes, it consisted of a high-speed general purpose computer (the fixed part *F*), which was to operate in conjunction with a second system (the variable *V*). See Figure 3.

The *F* computer was in his design to be the IBM 7090; the *V* was to be comprised of as many large and small high-speed substructures as necessary to carry out the defined set of special purpose problems. Furthermore, the *V* system would be reconfigured into whatever structure necessary to compute the class of special problems. The cooperation of the *F* and *V* systems would occur under the direction of a supervisory control unit (*SC*).

Reddi and Feustel approach the problem from a different perspective: the issue for them is the nature of von Neumann architecture, most specifically the implications of strict sequential code execution and the uniform internal representation of data [Red78]. While acknowledging the value of the von Neumann paradigm in the development of the technology, Reddi and Feustel saw sequential execution as an impediment to high speed computation and efficient resource utilization, because it does not exploit the parallelism inherent in a problem and in hardware structures.

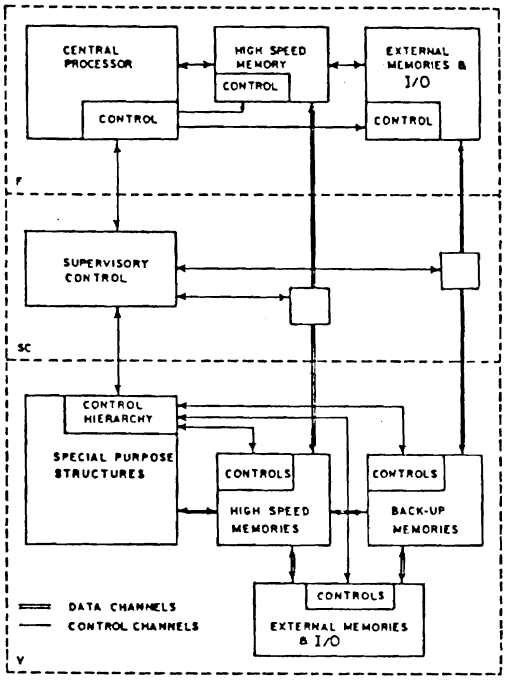

Figure 3 - Block Diagram of *V*, the Variable Structure Computer System

Of course, we can see this criticism as simply another version of Estrin's problem of practicable computability. The second characteristic of von Neumann architecture, the uniform internal representation of data, was seen by Reddi and Feustel as a problem when complex data structures were present. This was a special interest of Feustel, who had earlier developed the concept of a tagged architecture, which provided at the machine level bit structures that defined by type the data associated with them [Feu73].

Along with Estrin, Reddi and Feustel recognized that the solution of special purpose architectures, while enhancing performance for certain problem domains, also imposed a new version of rigidity on the computing environment. Their proposed solution was, like Estrin's, in the second, interconnection mode of Miller and Cocke, but it differed from Estrin's in that it recognized information flow rather than algorithmic structures. See Figure 4.

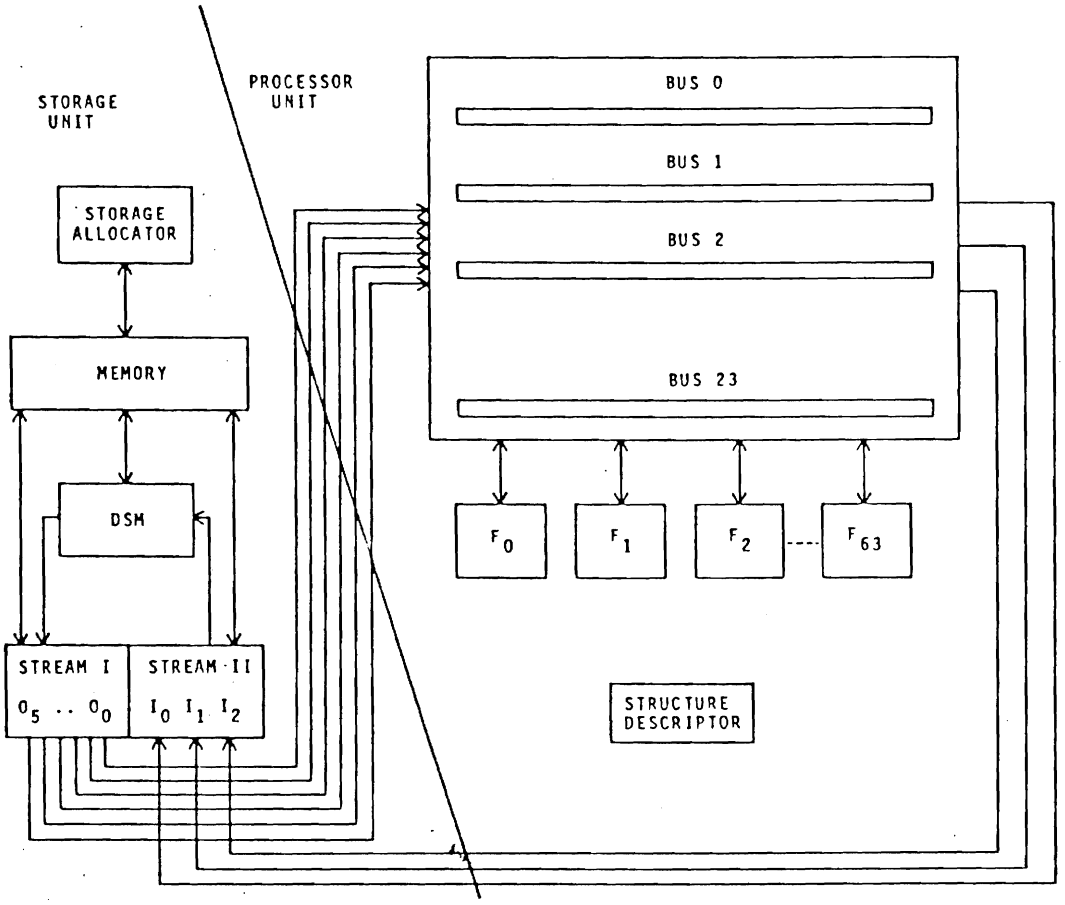

Figure 4 - The Restructurable System Architecture

The algorithm to be executed was to be compiled into program blocks, and the compiler would then establish a system configuration for each block. Reddi and Feustel's restructurable computer system made use of an intermediary language, Realist, which specified the configuration for each block. Rather than by an interconnection network, the configurations were to be implemented by bus units that were to provide data and control paths between resources. The system would support scalar operations as well as pipeline and parallel operations.

Thus we can see from these early proposed designs that reconfiguration does not arise from the initial theory of computation, but rather from the early attempts to enhance performance. These early attempts occur because the initial theory is seen to have been exhausted, or as Estrin saw it, basic computation theory does not coincide with the domain of practical computability.

3. RECONFIGURATION FOR FAULT TOLERANCE

3.1 Goals of this Discussion. Of the two major reasons for developments in reconfiguration, fault tolerance and performance enhancement, fault tolerance is the older concern, and there are strategies for fault tolerance that have little to do with reconfiguration, or that employ reconfiguration only in the widest sense. The function of the present section of this study is to clarify the definition of fault tolerance and the issues involved in it, and then to present a description and analysis of some of the major developments in architecture for fault tolerance. A comparison of fault tolerance to performance enhancement and their influence in design for reconfiguration will be held until the end of this entire study.

3.2 Defining Fault Tolerance. Siewiorek has well defined the issues involved in fault tolerance, and it is appropriate here to review his findings [Sie82] [Sie84]. We can approach his overall discussion of fault-tolerant architecture by constructing of tree, shown in Figure 5, based on his findings and pruned in the interests of reconfiguration.

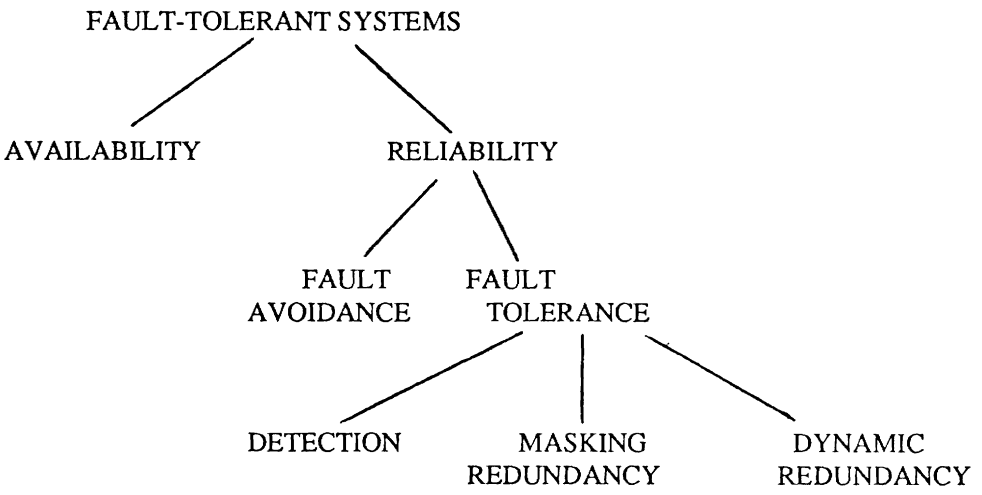

Figure 5 - A Tree of Fault Tolerance

Briefly, let us consider the nodes of this tree before we go on to focus on the node that interests us, here, which is the rightmost leaf, "dynamic redundancy." According to Siewiorek, fault-tolerant systems are either highly available or highly reliable. Availability is a function of time, *A(t)*, and expresses the probability that the system is operational at an instant of time *t*. If time goes to infinity, the function expresses the fraction of time that the system is available for useful computation. The availability of a system cannot be expressed as an unbroken linearity, of course: preventive maintainance and repair intrude on the time of availability. System reliability is also a function of time, *R(t)*. According to Siewiorek, it is the conditional probability that the system has survived the interval [0,*t*], given that it was operational at time $t = 0$. Reliability is a more critical issue than is availability, and is used to describe systems without online repair capability (such as in a satellite) or for which repair is impossible, either because of critical functioning (such as on an aircraft in flight) or prohibitive expense.

Reliability is provided either through fault avoidance or fault tolerance. Fault avoidance is conservative, and relies on the use of high-reliability components, component burn-in, and careful signal-path routing. It is important to notice the conservative thrust here: the goal is the prevention of failure. Thus, fault-tolerant systems can be seen as non-conservative, in that the goal is not the prevention of failure, but rather the manipulation of failure. Because failure is a state that is planned for -- we might say "built into" the system -- the design can be more adventurous. Failure manipulation is provided in all cases by redundancy, either time redundancy, usually provided by software, and basically characterized by repeated execution, or physical redundancy, most primitively characterized by the wheeling in of a new, duplicate system.

Siewiorek sees a redundant system as having up to ten stages -- fault confinement, fault detection, fault masking, retry, diagnosis, reconfiguration, recovery, restart, repair, and reintegration. He divides all of these stages into three classes, the three final nodes on the tree. Fault detection is actually a prelude to fault tolerance in this scheme; strictly speaking fault detection can occur as an end in itself, leading to a dead state of system

failure. In the present scheme, however, fault detection leads to either masking redundancy or dynamic redundancy; the tree above is therefore somewhat misrepresentative. Masking redundancy is, furthermore, not necessarily preceded by fault detection, and is not necessarily concerned with giving warning of failure or even detecting it. Multiple execution of the same algorithm, for example, with voting on results, is designed to mask failure, but will not give notification of failure.

The domain of interest in the present study is the rightmost node of the tree, dynamic redundancy, which is Siewiorek's term for what we call reconfiguration. It includes conditions of online repair following a combination of masking redundancy coupled with fault detection. It also includes the simple notion of switching whole systems. It is the most active, non-conservative of the strategies of fault tolerance, and demands further discussion. We might add that Siewiorek's conception stops in its development before the advances in design that unite fault tolerance and performance enhancement are encountered. These include multistage interconnection networks, and largely concern the problem of communication. Thus, the discussion of dynamic redundancy will be followed by a discussion of reconfiguration for performance enhancement, where we will perhaps see that "reconfiguration" is more fully developed, and where the term "dynamic redundancy" will not be appropriate.

3.3 Dynamic Redundancy. Lala conceives of a system with dynamic redundancy as one which has several modules, but only one operating at a given time; the others are standbys which will be switched in under an overall system strategy of fault detection and fault recovery [Lal85]. This accords with Siewiorek's scheme which begins with the simple notion of a complete backup system being manually substituted for the faulted system. A diagram of these developments is presented in Figure 6 [Sie82]. The first of these is the pre-1975 strategy of complete replacement. This strategy is clearly the simplest, although it is also the most expensive in terms of hardware. It is also the most manual, both in conception and in implementation. The second, the use of a switch to allow peripherals to be attached to either processor, limited the replacement strategy to critical components.

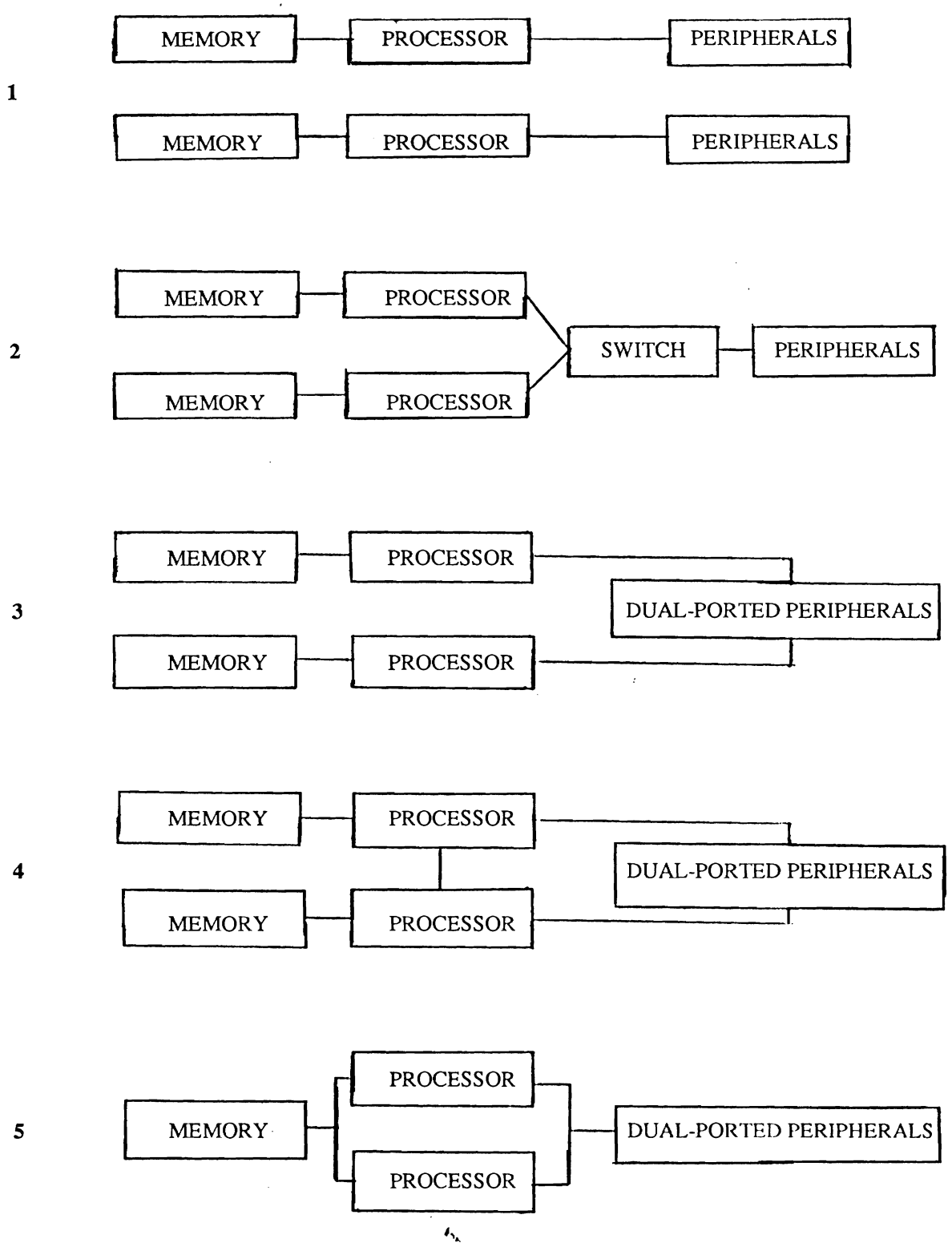

Figure 6 - Developments in Dynamic Redundancy

An improvement on the switching of peripherals, which was still a manual process, was the equiping of them with dual ports, as shown in number 3 of Figure 6. With the addition of an interprocessor communication bus, loosely coupled processing became possible. This is a major step away from the basic idea of having a second processor for the sake of standby only. One operating system could in normal functioning make use of both (or all) processors, and when a fault occurred the failed unit could be configured out of the system in a strategy of "graceful degradation." The final step in this evolution is the addition of shared memory to produce a tightly coupled multiprocessor. The processors share a common set of memory and peripherals, and under a single operating system any similar unit can back up a failed component. This last stage in the scheme of Figure 6 leads to the development of strategies to implement interconnection networks in fault-tolerant systems.

Before we investigate multistage interconnection networks, however, we should pause over two systems, the Tandem system and C.vmp, that represent respectively the last two stages of the development we have been discussing.

3.3.1 The Tandem NonStop system begins conceptually with strategy 1, which we have seen in Figure 6, in that the fundamental design principle is to duplicate everything, so that any single hardware fault will not prevent system failure. Tandem is a reconfigurable multiple processor system designed for online transaction processing [Kat78a]. However, the first advance over strategy 1 is that all maintenance and replacement of failed components is done online without bringing down the system. The second major advance, and the one that puts the Tandem system in the fourth category of Figure 6, is that the processor modules, of which there can be a maximum of sixteen, are all interconnected.

Each processor module consists of an instruction processor unit (IPU), memory, a bus control unit, and I/O channel, and a diagnostic data transreceiver (DDT). The presence of separate memory coupled with each IPU marks the Tandem system as representative of strategy 4 in Figure 6, rather than of strategy 5 in the figure. The IPU is a pipe-lined processor, and the module has up to 2 megabytes of storage, with a memory word width of

22 bits. The dual bus system that provides interprocessor communication which causes the Tandem system to be loosely coupled is called the DYNABUS. The buses are independent and separately controlled, and their power supply comes from different sources, so that a single power failure does not affect more than one processor. Messages are sent over the DYNABUS in 16-byte packets which are up to 32K bytes long. The I/O channel in each processor module has its own processor, which handles transfers between I/O devices and memory; this separate processing allows communication to proceed with limited intervention by the IPU.

The diagnostic data transreceiver (DDT), a part of each processor module, monitors the status of the other elements of the processor module, and reports any errors to the operations and service processor, which is an adjunct to the operating system. An example of the monitoring/reconfiguring capability of the system may be seen in the operation of the dual-port device controllers [Bar78]. I/O devices are connected to a given processor modules by one of the two ports of the controller, and the other one port is connected to another processor, but in normal function only in a standby capacity. When failure occurs, the DDT reports the failure, and the standby port is put into operation, thus allowing the completion of an I/O operation. Dual disk drives also allow a doubling of the data base, with automatic writing to both drives during normal operation, and a system of rewriting when a failed drive has restarted.

A copy of the Tandem operating system, called GUARDIAN, resides in each processor module. Again, the principle here is simple redundancy: a processor will always have a backup processor containing data and processing information which is refreshed at critical points; the presence of GUARDIAN in the backup processor allows that processor to proceed with operations should the first processor fail.

3.3.2 C.vmp. The final stage of the development modeled by Figure 6 can be demonstrated by the C.vmp system out of Carnegie-Mellon University. The system was originally designed in the mid seventies as the third of a series of machines with high

processor-to-memory bandwidth, all of which make use of commercially available hardware [Sie78]. C.vmp (for Computer, Voted MultiProcessor) had as part of its original purpose fault tolerance in an industrial environment, with electromagnetic noise, less knowledgeable users, and nonstop operation.

The response to fault-tolerance came in the form of a strategy for bus-level voting [Sie77]. As we can see from Figure 7, memory is separate from individual processors, and all memory/processor transactions must pass through the voting mechanism.

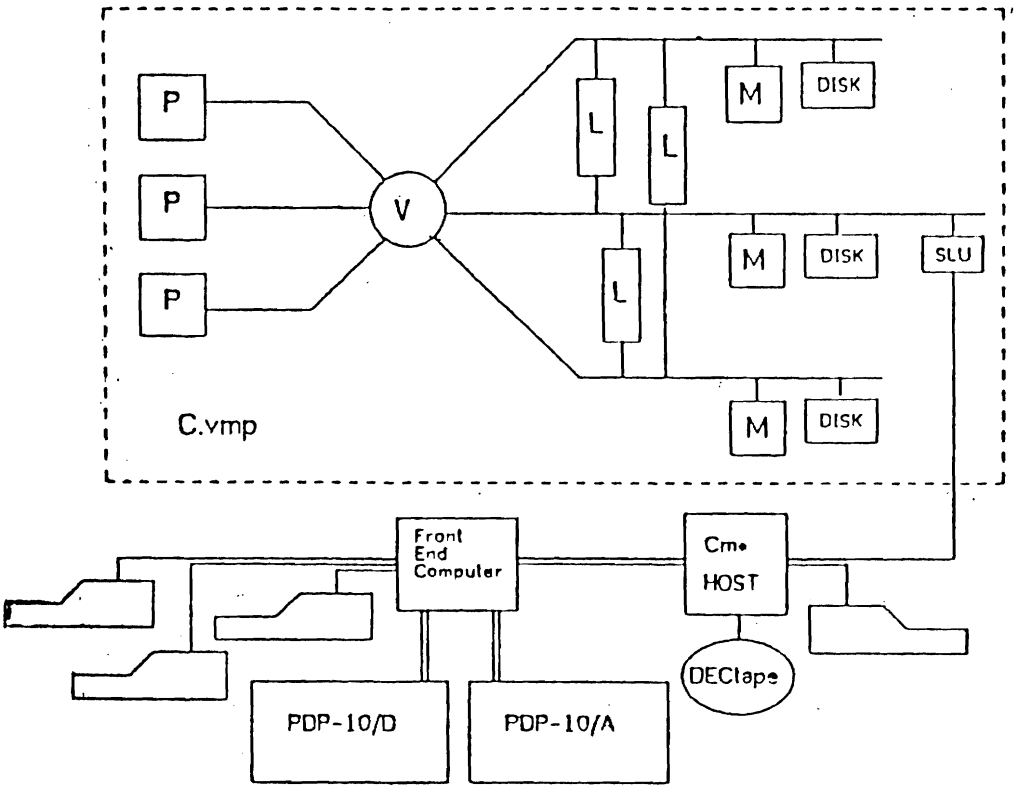

Figure 7 - C.vmp Voter-centered Architecture

The three processors can act individually, on different processes, and in this situation the voter is not activated. But when the processors are operating simulaneously on the same program, the voter is activated, either by an external event or under control of one of the processors. In this situation, what is basically a simple form of redundancy occurs: the processors establish results or request memory access that, when transmitting over the bus must compare with results from the other processors. Disagreements among the processors, which mean error, will prevent transmittal of information over the bus lines.

3.4 Fault Tolerance and Interconnection Networks. Many interconnection networks have been proposed, and they have been surveyed in, for example [Sie79a], [Mas79], [Fen81], and perhaps most fully in [Bro83]. It should be understood that while the term "interconnection network" can refer to any form of communication linking, including telephone systems, satellite networks, and manual switching of office equipment, the term is used here to mean multistage switching for very rapid data transfer among many processing elements in a limited environment under automatic control. This limitation of definition tends to be in agreement with common usage in the literature. It is also important to remember here that we have proposed that interconnection networks are the center of the stage for the development of fault tolerant systems that goes beyond the five-stage scheme proposed by Siewiorek and discussed above.

Feng describes [Fen81] the four fundamental decisions that go into the architecture of interconnection networks:

1) Operation mode, which can be either synchronous or asynchronous. Synchronous communication is demanded by data manipulation or data/instruction broadcast; asychronous communication is fundamental to multiprocessing, where connection requests are issued dynamically. A system can be designed to handled both synchronous and asynchronous communication.

2) Control strategy. The switching elements and interconnecting links establish communication paths by means of proper setting by the control unit. The two basic

methods of control are the use of a centralized controller and distributed control; in the latter method switches are set by individual controls.

3) Switching methodologies, of which there are two, circuit switching and packet switching. Circuit switching, which is appropriate for transmission of large amounts of data, establishes a complete physical path between source and destination, thereby tying up a considerable number of resources. Packet switching, which is appropriate for short data transmissions, establishes chunks, or packets, of data that are routed, essentially from node to node, without establishing all at once a physical path between source and destination. While interconnection networks tend to be developed for one or the other switching methodology, an interconnection network can be designed to implement both.

4) Network topology. The diagrammatic representation of a network that we most closely associate with the entire subject matter demonstrates the most obvious aspect of a network, its topology. Network topology can be most formally represented in graph theoretic structures of nodes and arcs, and it has been suggested that this form of diagrammatic representation is most suitable for meaningful analysis of network capability [Agr83]. Network topologies are of two kinds: static topology establishes passive connections between elements, with dedicated, nonreconfigurable links; dynamic topology establishes reconfigurable links controlled by active switching elements. Interconnection networks of the type under present investigation tend to be dynamic.

Interconnection networks are at the heart of the multiprocessing environment, and as we are presently seeing, they have become important in the development of fault-tolerant systems. Indeed, one of the themes of the present study is that interconnection networks provide the arena for the meeting of these two design issues. While many different interconnection networks have been proposed, they share similar characteristics, and Wu and Feng [Wu80] and Agrawal [Agr83] have shown that most of the proposed networks are topologically equivalent. Agrawal points to the value in this: initial design and fabrication of circuitry is expensive and production cost is low, which encourages the use of off-the-shelf components; therefore, if the circuitry designed for one interconnection

network is equivalent to that needed by another, the same off-the-shelf components can be used. Interconnection networks for different applications can be designed differently while still using the same components, and the control algorithms for different interconnection networks can be similarly applied [Agr83].

Three representative versions of this embellishment are now discussed, both for themselves and for the general principles they display. They are the Extra Stage Cube, the Gamma network, and the MPP, massively parallel processor, developed by NASA.

3.4.1 The Extra Stage Cube can be simply understood as an extension of the Generalized Cube that is presented elsewhere in the literature (e.g., [Sie81a], [Sie78b]), and that is analyzed in this study in the section on the PASM architecture, considered under reconfiguration for performance enhancement. It is a multistage cube-based network with N inputs and N outputs. It shares with other topologies of the multistage type the characteristics of $N = 2^n$ with $n = \log_2 N$ stages. Each stage has $N/2$ interchange boxes. Each of these interchange boxes has four legitimate states, straight, exchange, and lower and upper broadcast. The basic cube topology and the four states of the interchange boxes are illustrated in Figure 8.

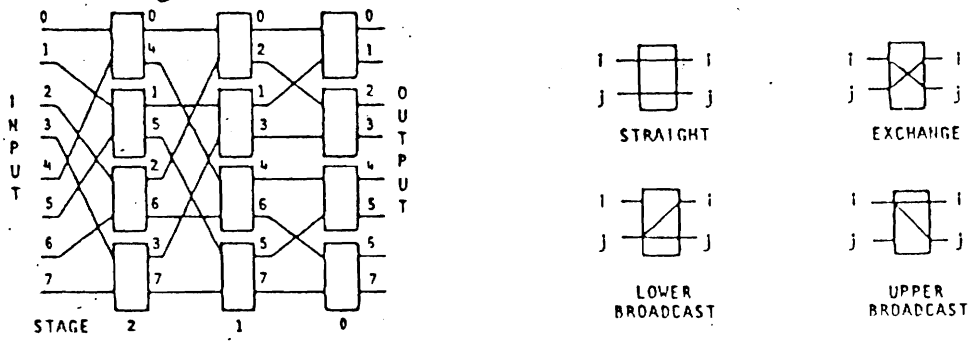

Figure 8 - The Generalized Cube and the States of an Interchange Box

The Extra Stage Cube is an extension of this basic design. An extra stage is added to the cube, as are multiplexers and demultiplexers. This extra stage is added to the input side of the network, and the multiplexers and demultiplexers are added to each end stage. This topology is illustrated in Figure 9.

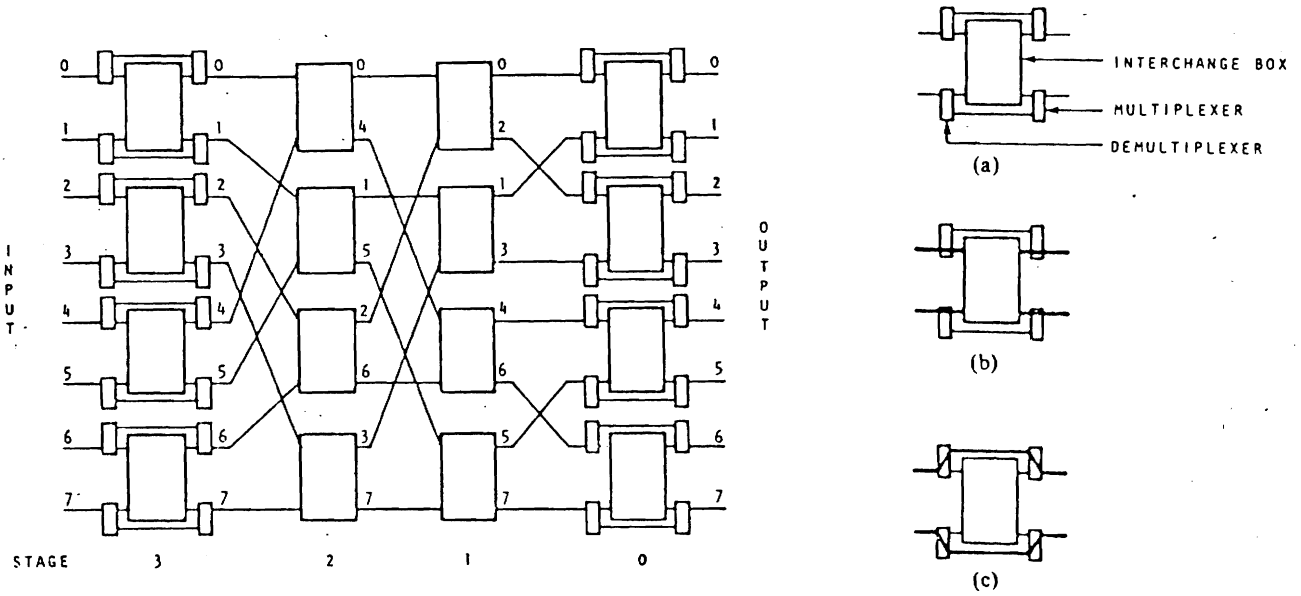

Figure 9 - The Extra Stage Network and the End-stage Switches

The function of the multiplexers and demultiplexers is to allow the end stages to be enabled and disabled, which is the basic mechanism for fault tolerance. We shall in this discussion refer to the extra stage as the leftmost stage and the final output stage as the rightmost stage. The leftmost or rightmost stage is enabled if its switches provide interconnection, and it is disabled if they are bypassed. The demultiplexer at each switch input and the multiplexer at each switch output, as shown in Figure 9, accomplish this task. And in the design of the Extra Stage Cube, whereas the switches themselves have individual controls, the multiplexers and demultiplexers of a given stage are set with one signal; thus the whole stage is either enabled or disabled.

Under normal, non-fault, conditions, the leftmost stage is disabled and the rightmost stage is enabled, which results in a working network that is identical to the Generalized Cube. If a fault is detected then reconfiguration occurs. If the fault is in the rightmost

stage then it is disabled and the leftmost stage is enabled. If the fault occurs in one of the middle stages then both leftmost and rightmost stages are enabled. A fault in the leftmost stage does not demand reconfiguration, because normal mode includes the disabling of that stage. And the routing for all of these contingencies is still based on the *i*th bit of the address of the output port to which data is sent [Sie79b]. Thus we have the principle of redundancy operating in an extended interconnection network.

3.4.2 The Gamma network demonstrates another strategy of redundancy for fault tolerance in an intercommunication network. Figure 10 shows the scheme of the Gamma network; a brief review of its workings will be given below.

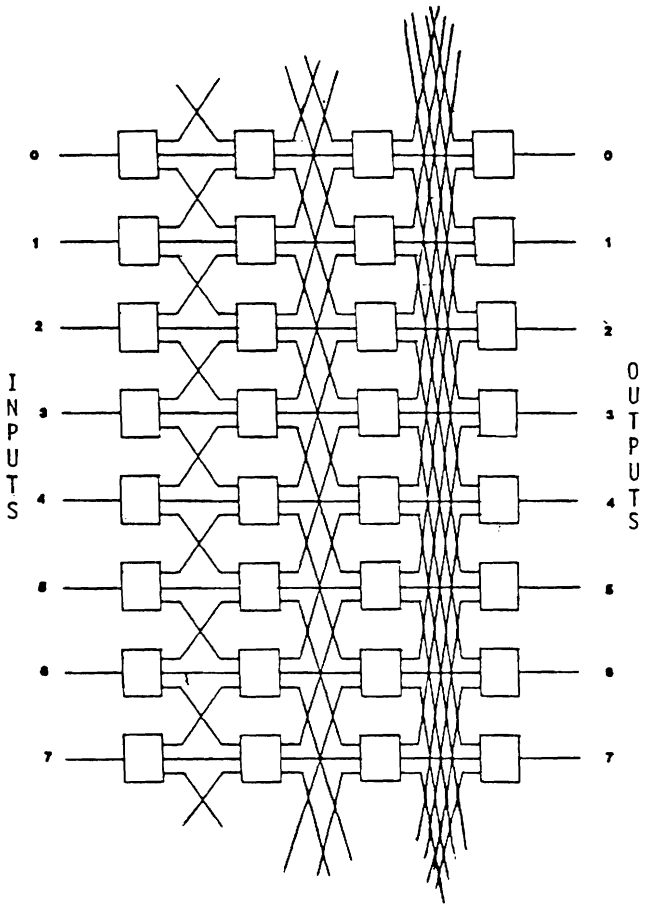

Figure 10 - The Gamma Network

The design is a refinement of the design for an inverse augmented data manipulation network (IADM) that comes from Siegel and associates [McM82a] [McM82b]. It has two main

innovative aspects: the network uses 3 x 3 switching elements, instead of the typical 2 x 2 elements, and it uses an elaborate "redundant number system" to represent and determine routing paths [Par84]. As we can see from the above figure, 3 input/3 output switches are used in the middle stage, with single input and output occuring in the end input and output stages. The three transition possibilities -- up, straight, and down -- work together with the redundant number system to produce multiple path possibilities for the exchanges.

The numbering system is redundant in the sense that values can have multiple representations, while still maintaining the same value. Digits in the numbering system can take three values, 1, 0, and 1, with this last value, 1, simply being a representation of -1 [Par82]. Thus, for example, the value 3 can be represented both as 011 and 101. Furthermore, there is a relationship between these three values and the three paths out of and into switches: each of the three values can represent one of the three switches.

With this association formed, the routing tag can be developed. The Gamma network has $n + 1$ stages with N switches in each stage, where $N = 2^n$. A message can change its route at n points in the system, and the routing tag is an n-digit fully redundant binary number. At each digit, therefore, the path up, straight, or down can be represented by the three numbers possible at the digit place. The various paths for the same source and destination result from using the difference modulo N of the source and destination, and by then representing this number in the redundant numbering system. Thus, if each stage is represented by each digit, and if each digit can be 1, 0, or 1, then by calculating the various representations of the difference modulo N of the source and destination, the different paths of the signal can be determined. The permutations that result provide possibilities than are more enhanced that Siegel's IADM network [Par84].

3.4.3 The MPP, massively parallel processor, was developed for processing satellite imagery at the NASA Goddard Space Flight Center [Bat80]. The system has a configuration of 128 x 128 microprocessors that can be used in parallel. Figure 11 shows a portion of the total array configuration.

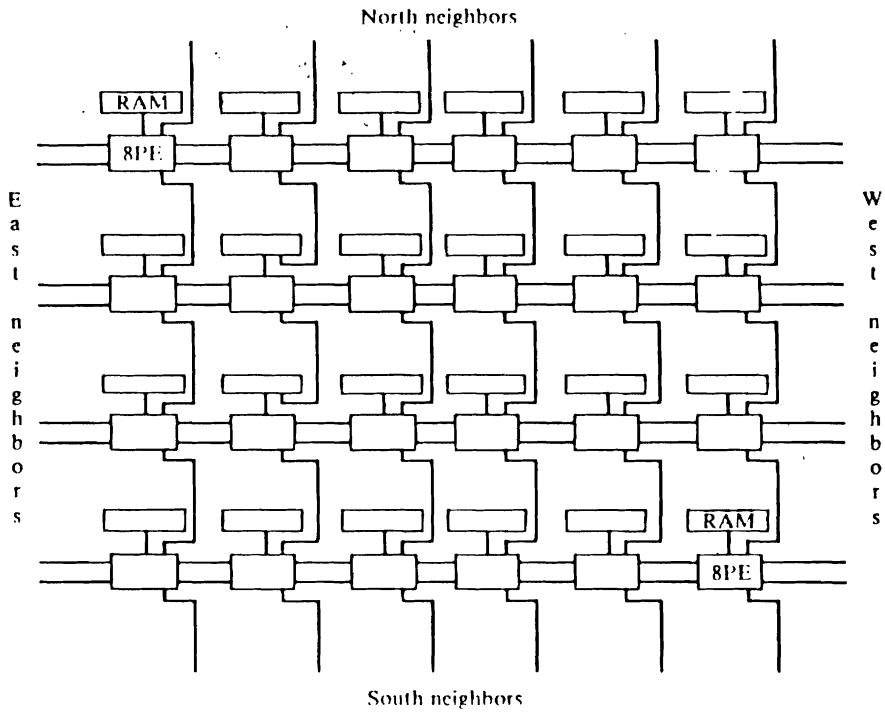

Figure 11 - A Section of the MPP Array

The MPP is essentially an two-dimensional array processor operating in SIMD mode, with each processor in the 128 x 128 configuration having a 1024-bit random access memory. The MPP performs bit-slice arithmetic with variable-length operands. Each processor element is connected to its nearest neighbors. The array topology can be explicitely rearranged into horizontal and vertical cylindersor into a torus. Figure 11 shows a portion of the total array configuration.

Failure in this massive system is controlled by having four columns of processors that are redundant to the main two-dimensional array, making the total configuration 132 columns by 128 rows. Circuitry is provided to mask out hardware faults; inoperative columns are simply bypassed, leaving a logical array structure of 128 x 128. The complexity resulting from the addition of the added elements is reduced by the necessity of providing interconnection along the rows of the array, not along the columns, since the substitutions are column based.

There are further complications to the MPP system, but this explanation reveals the basic method of redundancy that the network employs. A simple observation here is that this is quite a different scheme from others we have seen; it seems now appropriate to pause and offer some analysis of what we have seen in our investigation of reconfiguration for fault tolerance.

3.5 Summarizing Reconfiguration for Fault Tolerance. In this section we have attempted to define fault tolerance in general, and some of the strategies used to achieve it. Fault tolerance is an older concern than performance enhancement, as we are defining these terms and there are strategies for fault tolerance that have little to do with reconfiguration. The attempt has been made to clarify the definition of fault tolerance and the issues involved in it, and to present a description and analysis of some of the major developments in architecture for fault tolerance. Only in the last two stages of Siewiorek's scheme of a five-stage development toward "dynamic redundancy" can we begin to see what we call here reconfiguration. These last two stages were further discussed by an investigation of two specific systems, the Tandem computer and the C.vmp system, which are seen as representing the fourth and fifth stages of Siewiorek's scheme. This discussion of dynamic redundancy was therefore followed by a discussion of reconfiguration with ICN's, and "reconfiguration" is seen here as replacing "dynamic redundancy" when we begin to speak of the use of interconnection networks for fault tolerance. Investigation of the use of interconnection networks was demonstrated by three quite different designs, the Extra Stage Cube, the Gamma network and the MPP system.

The goal of reconfiguration for fault tolerance is not the prevention of failure, but rather the manipulation of failure. Because failure is a state that is planned for -- we might say "built into" the system -- the design can be more adventurous. In the early stage of fault tolerance, the tolerance is provided in all cases by redundancy, either time redundancy, usually provided by software, and basically characterized by repeated execution, or physical redundancy, most primitively characterized by the wheeling in of a new, duplicate system. However, while design in more advanced systems can be less conservative, and while fault

tolerance can become more accurate and efficient, the implementation of more recent fault tolerance does not replace the basic process of redundancy; it simply makes this fundamental process more sophisticated. The major shift is that the redundant elements are not purely redundant, in the sense of existing only for use in case of failure of other elements. Rather, they may have functions of their own which they perform while not being in what we might call the "redundant state." An adder that acts as a multiplier when the actual multiplier has failed is a simple example of this. In the non-redundant state it is an adder, and in the redundant state, entered when the multiplier has failed, it is a multiplier. And its goal remains the same: the correct execution of a specified algorithm in the presence of defects [Sie82]. But for our purposes, it is the place where fault tolerance links up with reconfiguration for performance enhancement.

4. RECONFIGURATION FOR PERFORMANCE ENHANCEMENT

4.1 Goals of this Discussion. Rather than attempting at the outset a theoretical model of performance enhancement and its relation to reconfiguration, in this section and the following two sections we will attempt to survey and analyze some of the more significant proposals for performance enhancement. Some of these system designs have reached fruition in the form of working machines, if only in prototype; others remain paperwork machines, which contribute in their own way to the development of thinking on reconfiguration for performance enhancement. A survey of some of these many proposals will first be attempted, in order to give the reader a sense of the range of ideas on the subject, and in order to provide a contrast to the proposals for reconfiguration for fault tolerance. In sections 5 and 6, the discussion will focus on the most sustained systems that have been proposed, not system by system, but under the two issues of communication and control. While we will not stop and deliberately contrast and compare the two sets of proposals, those for fault tolerance and those for performance enhancement, the relationship should be apparent, and will become the center of discussion in the conclusion of this study.

The developments in reconfiguration for performance enhancement include the dynamic architecture of the Kartashevs, PASM, the Star local network, and the NYU Ultracomputer. The dynamic architecture of the Kartashevs has developed over ten years and differs considerably from the others in communication, control, and other issues [Kar79a]. PASM (Partitionable SIMD/MIMD Machine), developed at Purdue University and at present in prototype stage of development, is a dynamically reconfigurable multimicroprocessor system [Sie81]. Star, a local computer network that is being designed to integrate image database management and image analysis into one system, gets its name from its topology: a star-connected communication subnet centralizes distributed-controlled switching elements to provide a tight coupling among a large number of autonomous elements [Wu82]. A recent entry in the field is the NYU Ultracomputer, which is a

general-purpose MIMD machine accessing a central shared memory via a message switching network with the geometry of an Omega-type network [Got83].

This analysis of designs should allow some final remarks on the nature of reconfiguration for performance enhancement. But first, it is necessary to provide some fundamental notions of what exactly "performance enhancement" means in the context of our discussion.

4.2 Defining Performance Enhancement. We might broadly define the development of computer technology, and thus the development of performance enhancement, as having four stages:

1) the machine-based technology, wherein the von Neumann design was fully developed and single-process operation control was left up to the programmer;

2) the operating system technology, which lifted the programmer away from the details that were common to all processes and placed them under the domain of the operating system;

3) multiprocessing, allowing for the use of the developed technology in pipeline and array processing;

4) reconfiguration, the stage that allows multiprocessing that is algorithm-driven, and that allows processing to conform to the manifold needs of an advanced, highly powered, high-demand environment, such as image processing.

While not always schematized in this manner, these developments are well known and fully presented in the literature. For our purposes, we should note that our concern with "performance enhancement" aligns with this fourth stage of development, which includes the concerns of parallel processing in both SIMD and MIMD modes, and that problem solving in the research usually centers on the communication links between processors and memory. Furthermore, we should observe that reconfiguration for performance enhancement, while making use of similar strategies, does not have the same concerns as reconfiguration for fault tolerance. However, the use of similar strategies in these two domains may provide the key to unification, at least in concept.

4.3 Early Developments. In a 1979 paper introducing basic principles of their own dynamic computer architecture, the Kartashevs review the major developments up until that time in reconfiguration design [Kar79a]. Their survey begins with the work of Estrin, whose work we investigated in Section 2 of this report. Estrin developed at UCLA in the late fifties and early sixties a "restructurable" system that pioneered the strategy of examining the algorithmic structure of a problem and then assigning the tasks of the problem to either "Fixed" or "Variable" subsets of the system [Est63]. This assignment was based on the pre-analysis of the problem and the subsequent "decomposition" of the problem into different tasks needing different architectures, two concepts fundamental to reconfiguration. The Kartashevs also mention the Illiac-IV computer, which allows the reconfiguration of one 64-bit processing element into two 32-bit or eight 8-bit processors; it is devised mainly for the enhancement of parallel execution [Bar68]. Other major work they discuss includes Lipovski's extension of the concept of a reconfigurable array processor developed for SIMD to the MIMD mode [Lip77], and the work of Reddi and Feustel, who like Estrin and others before them, proposed the matching of topology to alogrithm [Red78]. They introduced an intermediate language called REALIST, which identifies the structure appropriate to the computation needs, and they proposed the implementation of the system using APL. Clearly, at the point when the Kartashevs introduce their system much work had already been done.

It remains the purpose of the present section to survey some other developments, in order to extend the 1979 review by the Kartashevs, and to present the fundamentals issues that all proposals for reconfigurable architecture must face, as well as the various strategies that are possible.

4.4 The PM^4 System. This is an architecture out of Purdue University - the Purdue Multi-mode Multimicroprocessor system [Bri79]. Its development demonstrates the need for processing of images, an environment that is generally characterized as having massive amounts of data upon which the same relatively simple task must operate. A screen of 500

x 500 pixels of information from which basic texture analysis must be extracted is the obvious example. An SIMD machine is needed here. But the system should be reconfigurable, because this simple kind of operation is not the only need in image processing. The PM^4 system is designed to have three operation modes in addition to SIMD. In multiple SIMD mode, a number of SIMD operations can be executed in parallel. In MIMD mode, individual instruction streams have a sequence of scalar operations, and these parallel processes may be interdependent. Vector instructions may not appear in MIMD mode, but they may appear in the fourth mode of the PM^4 system, the Distributive Mixed Mode. Here, SIMD vector instructions and parallel MIMD processes are simultaneously executed.

(a) *Overview.* The system consists of N identical Processor-Memory Units (PMU), K identical Vector Control Units (VCU), a three-level hierarchical memory, and a set of interconnection networks and memory management units. See Figure 12. The three levels of memory are the local memory in both VCUs and PMUs, the shared memory with direct interconnection to the processors, and the lowest level, the file memory.

(b) *Vector control.* Each VCU consists of a microprocessor and a local memory (LM) and Local Memory Management Unit (LMMU). This local memory is part of the highest level of the three-level memory subsystem. The dominance of the VCUs in the design suggests that, in spite of the intention of having four modes in the architecture, the system is most strongly oriented to SIMD processing. Indeed, this mode is the one most carefully discussed in the proposal, and SIMD mode will therefore be the focus of discussion here. Vector control instructions and program of an SIMD process are loaded into the VCU local memory prior to execution. The VCU broadcasts instructions to all of the PMUs that have been assigned via reconfiguration to the given SIMD process. Disabling PMUs in the system that are not part of the reconfigured SIMD subsystem is a function of the VCU. There seems to be no particular tying of a given VCU to a given subset of PMUs

in the PM^4 system; if this is the case, then the system can be reconfigured in SIMD mode to utilize from 1 PMU to N PMUs.

(c) *Other Processors.* The Processor Memory Units, the PMUs, in the system resemble the VCUs in their organization. Like the VCUs, they consist of three units - a microprocessor, local memory (LM), and a memory management unit (LMMU). The LMs in the PMUs constitute the second part of the highest level of the memory in the system, the first part being the LMs of the VCUs discussed above. Each LM acts as a cache for its associated processor. The LMMU in each PMU loads and unloads local memory, and it also

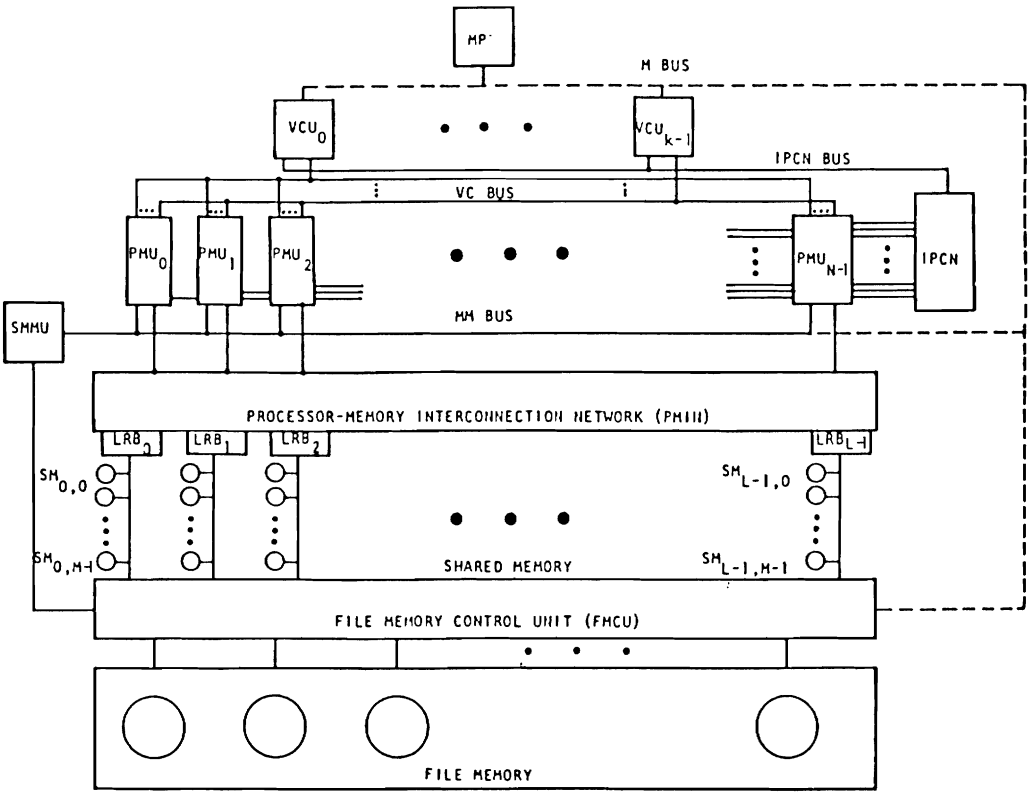

Figure 12 - The PM^4 Architecture

acts as a channel to transfer a block of shared memory to any VCU memory associated with the PMU in a reconfiguration. Program transfer from shared memory to the LM of the VCU does not have to pass through the processor in a given PMU; rather a multiplexor connects each PMU with the Vector Control busses to the LMMU or the processor. Access can therefore be through the LMMU. The multiplexor also broadcasts instructions for a VCU to logically connected PMUs in SIMD mode.

(d) *Interconnection Networks.* Figure 12 indicates the presence of four communication subsystems in PM^4 : between the VCUs and the PMUs, the interprocessor communication network (IPCN), the processor to shared-memory interconnection network (PMIN), and the connection to the file-memory control unit (FMCU).

VCU-PMU communication and the IPCN are the links of most interest to the problem of reconfiguration, and they will therefore be the focus of this brief discussion. Fundamental control of reconfiguration for SIMD mode during VCU-PMU communication resides in the VCU, in that the given VCU broadcasts instructions to its subset of PMUs. The VCU is also capable of sending permutation function commands to the IPCN for the purpose of permuting the data in a group of PMUs. The VCU also has the ability to mask out PMUs, which allows the VCU control over the broadcasting of instructions; it can thus change the configuration of its subset in SIMD mode. The IPCN, also of interest in reconfiguration strategies, was not fully worked out at the time of the initial proposal [Bri79], but its major purposes are clear. Partitioning of the network, which can occur only in fixed-sized blocks, is to be implemented by the IPCN, in order to allow parallel execution of small-size SIMD operations. It is also used to implement permutation functions needed for SIMD processes. The data from multiple SIMD processes can be permuted under control of the IPCN.

4.4 The CHiP Computer. More than other designs, the CHiP (*C*onfigurable, *Hi*ghly *P*arallel) computer takes into consideration the implications of VLSI technology [Sny82].

For one thing, none of the communication strategies in the design makes use of crossover paths, which have been demonstrated to decrease efficiency and increase cost when implemented on a chip [Fra81]. And the design starts from the developments in what are referred to in the proposal as "algorithmically specialized processors," which are architectures designed for processing of particular problems, such as systems of linear equations, tree processing, searching and sorting, and data base querying. The CHiP architecture grapples with the rigidity inherent in these different designs not by interconnecting a set of dedicated processors, but by implementing all of them - or most of them - in one lattice design of switches and processors. It exploits implications of "algorithmically specialized" processors, including construction based on a few easily tessellated processing elements, locality of data movement, and the appropriateness of pipelining. Clearly the purpose of reconfiguration here is quite different from what we saw in the PM^4 design. There, reconfiguration allowed implementation of SIMD, MSIMD and MIMD processing in image processing; here, the goal is more multi-purpose, and reconfiguration allows efficient use as well as parallel processing. It is particularly suited for computationally dense processing, for example, solving a system of linear equations [Gan81].

(a) *Overview.* The machine consists of three parts: a group of identical microprocessors, a switch lattice, and a controller. The switch lattice, a regular structure formed from programmable switches connected by data paths, is the innovative aspect of the design. The microprocessors are connected in a regular pattern to the switches, and the connection of the two groups of units form the overall lattice structure. The switches have local memory and can store several configuration settings. Using circuit switching and the implications of the interconnections, the switches set static connections in the mesh of possible paths. As can be seen from Figure 13, different patterns of switch-processor interconnection are possible. Part of the goal in implementing the architecture is to have as much of a lattice as possible placed on one chip, and, as mentioned above, the design,

while intricate, will never involve crossover paths, and therefore is appropriate for wafer-level technology.

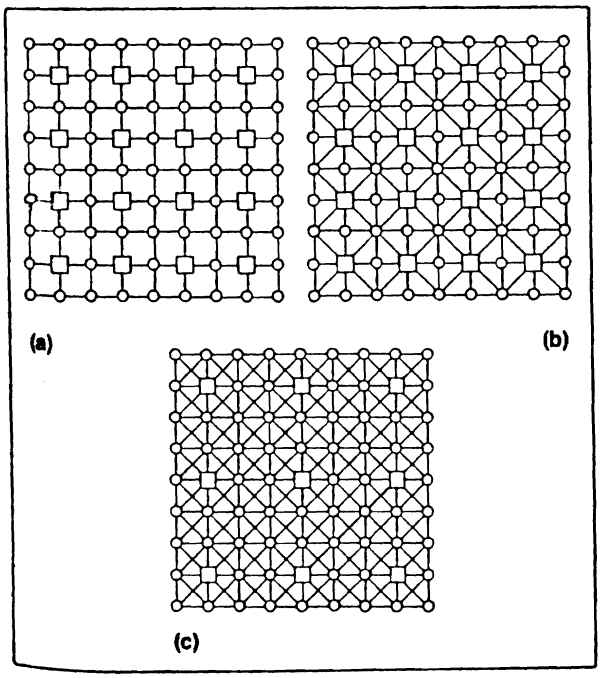

Figure 13 - Three Lattice Structures in CHiP

For a given process, demanding a given architectural pattern, the lattice is reconfigured: a subset of the overall group of switches and processors is activated to create an algorithmically specialized processor. Switches contain local memory that stores configuration settings. Direct, static connections are established between processors, and these connections are maintained until the task connected with this architecture is completed. Figure 14 shows reconfiguration into a mesh pattern; Figure 15 shows reconfiguration for binary tree processing. Note that the goal here is not partitioning for the sake of creating simultaneously operating subsets, in that only one subset is created at one time. Therefore, parallel processing lies beyond the domain of the fundamental design.

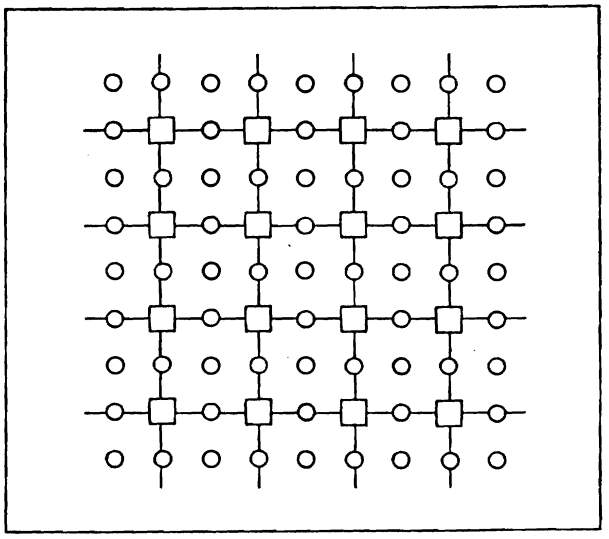

Figure 14 - The Switch Lattice Configured as a Mesh Pattern

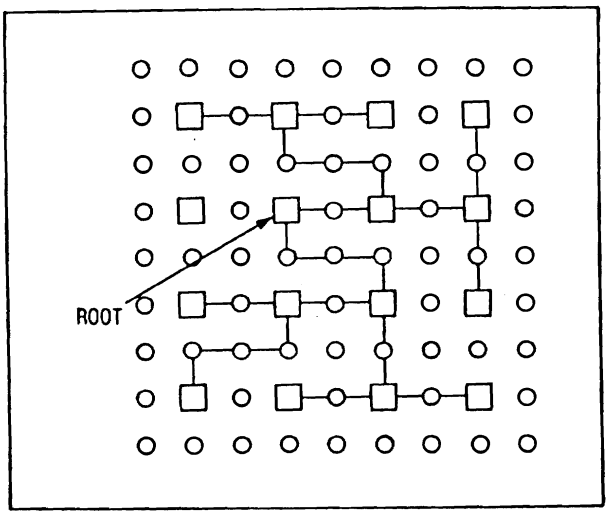

Figure 15 - The Switch Lattice Configured as a Binary Tree

(b) *Control.* Switch memories are loaded with configuration settings by the controller, using a separate interconnection network. The settings for a given configuration must be loaded into the same memory location in each switch. This loading occurs before processing, and is performed in parallel with the processor program memory loading. The memory locations must be the same in all switches, partly because the controller is operating in broadcast mode when it sets the switches. The setting remains static throughout processing in a given configuration. When a new configuration is necessary for the next phase of processing, the controller again broadcasts a switch setting message. There is thus only one logical step in reconfiguration before processing resumes.

(c) *Switches, lattices, and the interconnection patterns.* The various possible lattice patterns in Figure 13 demonstrate that switches can have two different relations to the processors: they can stand alone as the connection between two processors, or they can be

part of a set of switches forming a corridor. This allows specialization of switch use, with corridor switches tending to perform routing, and "coupling" switches acting like processor ports for connection with corridor transmission. Lattices themselves can also take different forms. Fewer switch corridors provide tighter coupling but allow for less flexibility and a potentially high incident of processor underuse. Maximum efficiency finally depends on the particular applications of the system. And final patterns of embedding do not depend on geometry alone; more sophisticated methods of use need to be employed.

4.5 TRAC. The Texas Reconfigurable Array Computer, developed at the University of Texas at Austin, was originally designed for scientific processing, but the design demonstrates a common goal of reconfigurable architecture - the restructuring of one system for a wide range of use. The focus of its design innovation is its dynamically reconfigurable banyan network [Sej80]. Of the systems we are discussing in this section, it is closest to the CHiP computer in intention - a multi-use system - yet it stands out in its focus on intercommunications needs. While it is no longer under development, its design proposal allows us to see a certain type of strategy in reconfiguration: interconnection of many system elements for the sake of various tasks.

(a) *Overview.* The initial TRAC design calls for a system connecting 16 processors to 81 memory and I/O elements. The resources can be partitioned into from 1 to 16 units, which run independently. As with other designs, independent control of partitions and real-time (referred to in TRAC literature as "space sharing") rather than time sharing are goals. The system is dynamically reconfigurable while running.

The TRAC subsystems can operate in various types of parallel execution. During asynchronous MIMD operation, a given task may fork into subtasks. The system also supports asynchronous pipelining. Vector parallelism is also supported, as well as synchronous parallelism with external control of startups and interrupts.

(b) *Control.* Control centers in the scheduler. When a task begins, it passes

information to the scheduler about type of data structure and the urgency of the task. Urgency can determine the number of processors allocated. The scheduler acts as arbitrator among tasks for resource contention. A special aspect of the system is the concept of "folding" of elements in a vector. If a task is allocated fewer processors than it needs, elements are packed into the available memory modules, in a process that doubles up the use of the available memory elements This packing is transparent to the user, and does not require additional machine-language instructions.

(c) *Processors.* Each processor operates with 8-bit operands, and multi-precision data is processed in parallel using multiple processors. An instruction tree connects all processors in a partition during an instruction-fetch cycle. The memory element of one of the processors fetches the instruction then broadcasts it to all of the other processors in the partition.

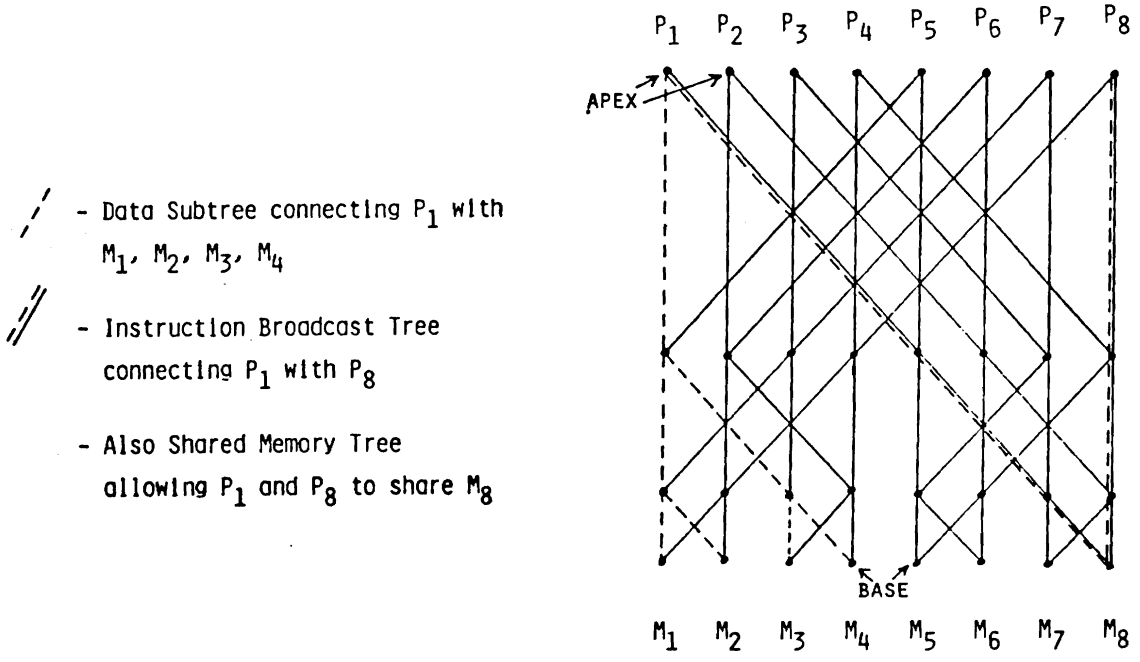

Figure 16 - The Banyan Interconnection Network for TRAC

(d) *The banyan interconnection network.* At the heart of the TRAC system is its banyan network. Three types of "subtrees" in the network are established in the system:

data trees, instruction trees, and shared memory trees. They are trees in terms of the utilization of the banyan configuration (see Figure 16) but they perform logically as busses. The data tree connects a processor with memory; the instruction tree broadcasts instructions to participating processors in SIMD mode; shared memory trees connect a set of processors to a single memory module for the purpose of sharing data. The banyan configuration is found to be attractive for the reason that most designers find multistage interconnection networks attractive: the decreased number of switches. Unlike the crossbar networks, the switch number of which increases $O(n^2)$, the banyan network switch need increases $O(n^*\log n)$.

4.6 Other Proposals. Many other reconfigurable architectures have been proposed, and have attained various stages of development. Lundstrom and Barnes describe a system to be used as a Flow Model Processor in the Numerical Aerodynamic Simulator for NASA [Lun80]. Its prime interest is MIMD for parallel processing. The system includes memory that is connected individually to each processor and memory that is shared; the goal is maximum memory availability to reduce conflict. The interconnection network chosen to connect the proposed 512 processor/local-memory with shared memory is the baseline network of Wu and Feng [Wu78]. Reconfiguration is explicit, with source code that compiles into the same program for execution for all processes in an array. Use of Fortran is proposed, with an extension of two new instructions, the concurrency construct "DOALL" and the definition of index sets through "DOMAIN," a means for distinguishing local from global variables. All processors can request connection to any memory module in the 512- processor x 512-memory configuration. In another paper, Gray expands on Snyder's work on the CHiP system to offer a distributed control structure that can be used to grow automatically the configurations described in CHiP from seed states implanted at arbitrary locations in the array [Gra82]. This is an enhancement to the Snyder design, in that the seed states replace the need for setting the switches individually and externally. (See section 4.4 of this study.) Based on the assumption that the different possible

configurations of the lattice are fixed, predetermined, and capable of being stored locally in the memory of the selected "seed state" switches, patterns of configuration are generated outward from the "seed state" switch to the neighboring switches. This reconfiguration strategy is aimed at functional enhancement but also fault tolerance. All processors are identical and control is distributed throughout the array, and, as in the CHiP architecture, no multistage interconnection network is implemented.

A reconfigurable multimicroprocessor research system under development at Los Alamos National Laboratory is reported on by Trujillo [Tru82]. It is a tightly-coupled, shared-memory MIMD system supporting reconfiguration between processors and memory nodes, for the purpose of structuring processors into rings, trees and stars. It uses a full crossbar, multiple bus network between processors and memory to allow for full processor-to-processor and processor-to-memory communication. Three types of processors are included in the system: a system control processor, general floating point processors, and dedicated data transfer processors. Processor-to-processor communication is implemented indirectly through the processor-memory interconnection by data transfer processors that move data between global memory nodes. Processor-to-memory communication is provided by memory-mapping logic at each processor, a multiported memory controller at each global memory node, and the multiple bus interconnection network. An orthogonal packaging scheme allows minimal bus lengths for the physical connection of processors and memory nodes. The system is designed as a research tool for implementing and evaluating parallel processing algorithms on different multiprocessor architectures to be reconfigured as subsets. A different strategy is the data-flow, "language-based" reconfigurable architecture proposed by Chen and Ritter that is designed for use as a processor for parallel computation of variable image neighborhood operations [Che84]. Reconfiguration is important here because the data of pixel neighborhoods is variable. The system is "language-based" in that processing is defined in terms of a few elementary operations and functions; various image processing tasks, such as edge detection and Fourier transformations, are developed out of the

elementary operations and functions. The tasks are then expressed as data flow graphs that are mapped to the reconfigurable system. Image data is input through a front-end system that interfaces with a distributed network that leads to various operation modules. Reconfiguration is controlled by an arbitration network.

A methodology for performance enhancement through reconfiguration architecture for VLSI design comes from Japan [Iwa85]. The increased numbers of integrated circuits that can be put on a chip also means increased design manpower and design time. What is suggested is a hierarchical design structure, to distribute tasks in the design process, and versatility of the inner modules, to allow for multipurpose use. A hard disk controller that can interface with many different drivers and that can be programmed by users for such variables as track format and parity byte length is the first implementation of the method. Finally, the Cosmic Cube, an experimental computer for highly parallel processing, has been developed at Caltech [Sei85]. See Figure 17.

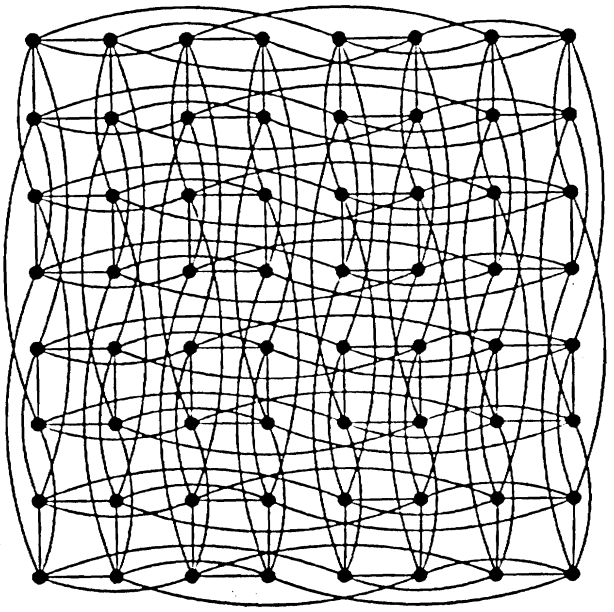

Figure 17 - A Hypercube With 64 Nodes

The Hypercube consists of 64 small computers that are connected with bidirectional, asynchronous, point-to-point communication channels. This is quite different from other proposals, in two major ways: 1) the MIMD machine uses message passing rather than

shared variables, and 2) the processor/memory units, which do not need a interconnection for processor/memory access, are all connected in a "hypercube" mesh that allows one-to-one communication between processors. A direct network like the hypercube is intended to work very well with large numbers of nodes. The major implication of the point-to-point communication in the Hypercube is that there are no switching mechanisms, and the processor and storage units are ideally intended to reside in high-density packaging, most ideally on a single chip.

This review of various architectures should demonstrate the range of goals and designs that use reconfiguration strategies for performance enhancement. The next two sections of this paper will focus more in depth on the two issues of communication and control in four major systems.

5. STRATEGIES FOR INTERCONNECTION

Interconnection directly influences processor/memory relationships and determines use of local versus shared memory [Gaj85]. The distinction has been made between "logically partitioned" systems - those that use software techniques - and "physically partitioned" systems - those that use hardware switches [Sie79b]. If we use this distinction, then we are speaking here of physically partitioned systems, although software control is present. The various strategies proposed for interconnection always have speed and cost as issues, but, as we shall see, changing technology is also an issue, and it may well alter the speed and cost of a given strategy.

5.1 The dynamic architecture of the Kartashevs makes use of the simplest reconfiguration strategy of the four under analysis. The initial proposal calls for a lining up of computer elements, CEs, each containing a processor and local memory, and connecting them with a data path from one to the next [Kar79a]. That is, if there are five CEs, $CE_{1\text{-}5}$, CE_1 can be connected to CE_2, but not to CE_3, and so forth. See Figure 18. The connecting lines (MSEs) can assume three modes: right transfer, left transfer, and no transfer. If transfer mode, left or right, is in operation, then the adjacent CEs in question are linked, or are part of a subset computer C. In Figure 18, $CEs_{1\text{-}4}$ constitute a subset, and the MSEs between them are in transfer mode. The MSE between CE_4 and $_5$ is in no transfer mode.

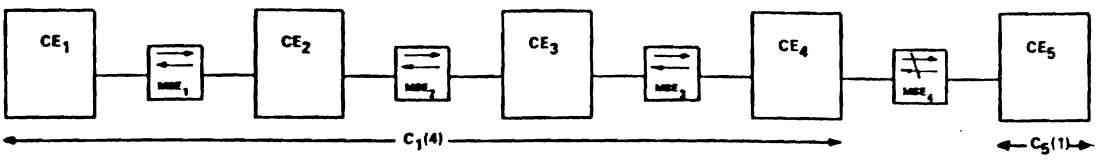

Figure 18 - DC Group with Four Processors Connected

Further notation is necessary here. CEs are linked together to form a subset, or "computer," C. Each C has k number of CEs, and one of those CEs, i, is the leftmost, or most significant, in the linear, horizontal configuration. Thus each "computer" is designated as $C_i(k)$, in Figure 18, the "computer" interconnected by the MSEs in transfer mode is $C_1(4)$. This notation points up the limited configuration possibilities in the Kartashev system: only adjacent CEs can be connected. The different possible configurations therefore is quite limited, and easy to determine. A five CE system, for example, yields only $C_1(5)$, $C_1(4)\,C_5(1)$ ... $C_1(1)C_2(1)C_3(3)C_4(4)C_5(5)$.

In a later paper [Kar80a] refinements were made to the original proposal, to loosen the tight coupling between processors and local memory elements. Basically, interface units are introduced into the design to allow each processor to communicate with all or any of the memory elements, not just the one that was tied to it in the original proposal. However, the limitation of communication only between adjacent processors, and the resulting limited set of configuration possibilities, remains; more recent work on task pre-analysis [Kar82a], and the most recent discussion of the overall system [Kar86], retain the basic elements of the original design.

This proposed reconfiguration strategy has the advantages of simplicity and fast data transfer rate. And in an implementation with many processors, there would be considerable performance improvement over more rigid systems [Kar78a]. However, the intercommunication structure, based on connection of adjacent processors only, is the least versatile of the structures we are investigating, and clearly, in an ongoing processing environment, the loss of performance due to fragmentation will be great.

5.2 The PASM architecture, when first fully proposed [Sie81a], did not have a specified interconnection network; two different possibilities were being considered, the Generalized Cube and the Augmented Data Manipulator (ADM). Recent publication on the project [Sch86] suggests that the decision has been made to implement a multistage cube network. The

goals, for whatever network, are the same: 1) a switch growth rate that is less than the N^2 growth rate of crossbar, the Cube having N/2 switches and the ADM N switches; 2) distributed control by routing tags generated by each processor; 3) SIMD and MIMD operation; and 4) partitioning into independent subnetworks [Sie81a].

The interconnection network is to be used in PASM to connect processor/memory elements (PEs), and the goals for the network parallel the goals for the system at large: 1) massive processing, to the size of 1024 processors, which demands a reduction in the number of switching elements; 2) total reconfiguration potential for the processors, which can only be attained through distributed control; 3) application to all necessary tasks for image processing, which demands both SIMD and MIMD; and 4) potentially total control in subnetworks. In SIMD mode, the machine consists of a control unit, PEs, and the interconnection network. The control units broadcast instructions to the processors; and whatever subset of processors has been grouped, and whose data paths to the control unit have therefore been enabled, execute the same instruction at the same time. Data is taken from the local memory associated with each processor. In MIMD mode each processor can follow an independent instruction stream, with instructions coming from the individual memory associated with each processor. Here the controller does not broadcast instructions, but it may coordinate processor activity.

The Cube network has been presented in the PASM literature under at least three different names, "Generalized Cube" [Sie81a], "Multistage Cube" [Sie80], and "Extra Stage Cube" [Ada82] [Kue85b]. This leads to some confusion, so the present discussion will be oriented to the basic design of the Binary *n*-Cube network, designed by Pease [Pea77]. See Figure 19.

The Binary *n*-Cube network is appropriate to PASM because it was originally designed for processor-to-processor communication rather than for aligning data between memory and processors [Bro83]. The Cube is somewhat analogous to the Omega network,

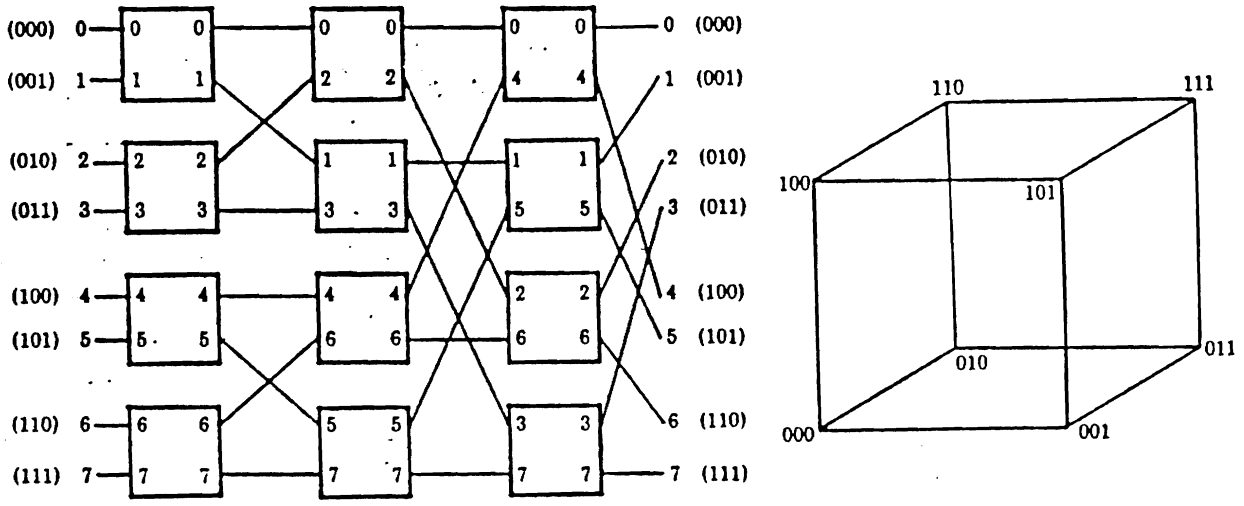

Figure 19 - The Cube Network, in Topology and Cube Transformation

but the difference is shown by the graphic representation of routing along the edges of a three-dimensional cube in *n*-space. Horizontal lines connect points whose labels differ in the low-order bit position, diagonal lines connect points whose labels differ in the middle position, and vertical lines connect points with differences in the high-order position. Mapping these connections to the multistage network represents the strategy for individual box control: the addresses of the two input lines to an interchange box at stage *i* differ only in the *i*th position [Sie79a]. The elegance of Siegel's proposal lies in the use of the cube structure to partition the set of connected elements into subsets that constitute independent networks [Sie80]. Reconfiguration is greatly enhanced, clearly, over the linear strategy of the Kartashevs. The number of permutations is greater; however, blocking still occurs, both in the set and in the subsets.

5.3 The Star local network is the only system under analysis that takes into consideration in its communications strategies the ISO/OSI seven-level reference model [Zim80]. Star is designed for image processing; it organizes multiple host computers, VLSI units, memory units for real-time image analysis, and large-scale database management units around the communication subnet Starnet [Wu82]. This subnet implements the first three levels of the OSI model, that are normally referred to in the literature as the physical, datalink, and network layers. Star is the most loosely linked system of those we are studying.

Star makes use of a modified baseline network. A baseline network unmodified displays characteristics similar to those employed in PASM: it provides multistage connection between elements, and it expands at a growth rate less than that of the crossbar. But as we have seen, these multistage networks allow for only one path between elements and a high blocking rate. Thus, the modification to the baseline network proposed in Star is the addition of an extra stage, as shown in Figure 20. The goal here is to provide greater fault tolerance and higher availability. Simple analysis of Figure 20 reveals that the extra stage allows the network to have two connection paths for each pair of elements. The routing scheme stays the same except for the extra stage, which is the new first stage. Both outputs of the source switching element - which is the new stage - will lead to the destination; thus selection can occur at the source based on priority or system fault.

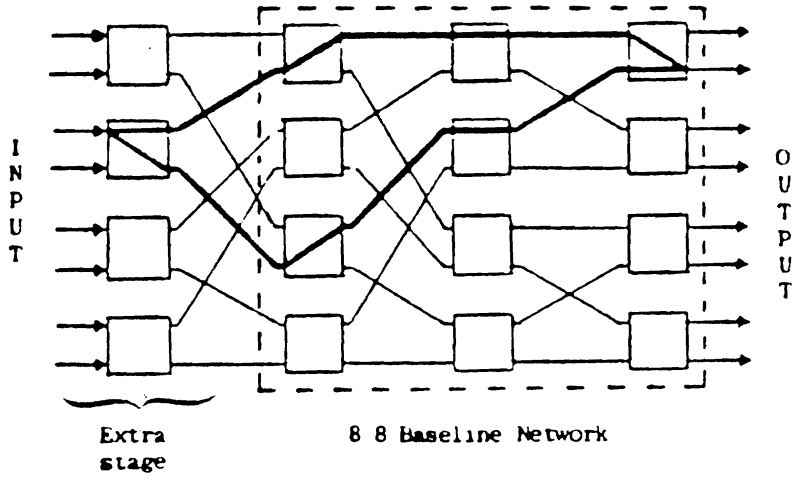

Figure 20 - Star's Modified Baseline Network

5.4 The NYU Ultracomputer uses reconfiguration of its network for support of a message-passing strategy; this purpose puts the design outside the general realm that we are discussing here, which is reconfiguration for the purpose of the creation of system partitioning for the sake of fault tolerance and/or performance enhancement. However, its design of a shared-memory, multiple-instruction-stream, multiple-data-stream sytstem includes interesting variations on our present discussion of interconnection strategies, and therefore a review of the system seems warranted.

The Ultracomputer and its interconnection network can be described in the context of its goal to approach the "idealized" parallel processor, for enhancements to the network make that goal possible [Got83a]. The ideal parallel processor consists of autonomous processing elements sharing a central memory; however the crucial issue is the possibility of simultaneous reads and writes directed at the same memory cell and accomplished in a single cycle. The designers acknowledge the physical impossibility here, and offer instead of a "real" parallel processor offer a "virtual," as we might call it, version of the real thing. This is accomplished through a single primitive, the fetch-and-add operation.

Behind this operation is the "serialization principle," which in a sense is a rewriting of the very notion of parallelism. The principle is that the effect of parallel processing can be seen as a serialized, unspecified, order of operations. A simultaneous request to the same memory cell for one load and two stores, for example, results is what can be seen as a serial process. The memory cell comes to contain one of the quantities written to it, but not both, and the load will return either the original value or one of the stored values; and because there are two different stores, even if a stored value is returned it is not necessarily the one that the memory cell finally contains. All of this is accomplished in one cycle, not a series of cycles; the serialization principle describes effect, not implementation.

The function of the fetch-and-add operation is to implement the serialization principle. The operation appears as F & A (V, e). V is an integer variable and e is an integer expression, and the operation is indivisible. The operation returns the old value of V and replaces it in memory by the sum of $V + e$. That is, two operations that we would normally consider to be separate, and potentially conflicting, are put in one "critical section" unit. The serialization principle is in operation here in that if V is a shared variable and many fetch-and-add operations address V simultaneously, they would appear as if they had occurred in an unspecified order; that is, each operation will yield an intermediate, and different, value for V and the final V stored in memory would be a result of all operations. This includes the possibility of the various fetches having arbitrary

results. If PE_i executes ANS_i <-- F&A (V, e_i) and simultaneously PE_j executes ANS_j <-- F&A (V, e_{ij}), and if V is not simultaneously updated by yet another processor, then, in addition to V in memory becoming $V + e_i + e_j$, one of two conditions will occur with the fetches:

ANS_i <-- V and ANS_j <-- $V + e_i$

or

ANS_j <-- V and ANS_i <-- $V + e_j$.

And always, V <-- V +e_i + e_j. The goal is the processing of parallel algorithms without critical sections, exclusive of the fetch-and-add instruction, and some results of this execution in the Ultra environment have been reported [Kru82]. All of this takes place in the context of an interconnection network that basically makes use of the Omega topology pictured in Figure 21.

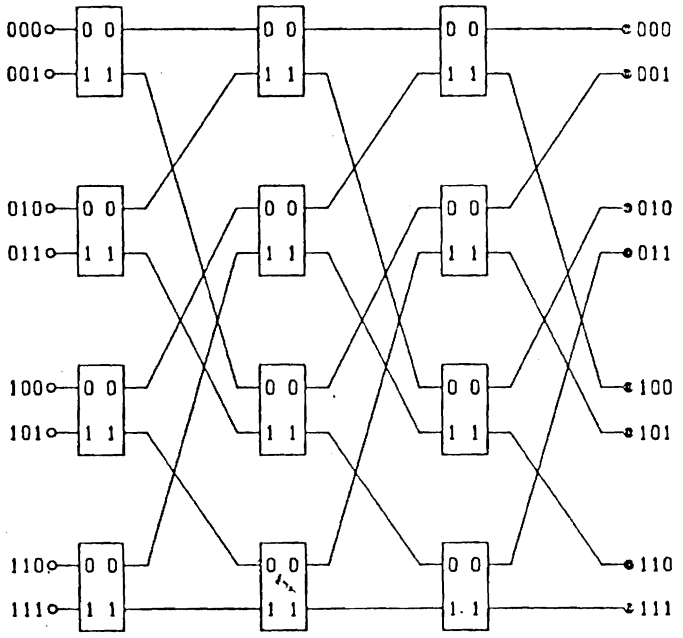

Figure 21 - The Ultracomputer's Omega Network

The nature of reconfiguration in the Ultracomputer resides here: the network uses a sophisticated switching design to allow the system to approach the ideal parallel processor described above. This is only in a limited sense reconfiguration for performance enhancement in the sense that we are in general discussing; for one thing, in no way does the reconfiguration of the Ultra network change the topology of the system. The goals for the network include three that it shares with other users of this kind of network: 1) bandwidth linear in N, the number of PEs; 2) Memory access time logarithmic in N; and 3) expansion at a rate of N log N. However, it has two special goals: 1) routing is to be performed at the switch level; and 2) concurrent access by different processors to the same memory cell occurs in the same time as access by one processor. The two special goals are associated with the issues involved with the serialization principle, the fetch-and-add operation, and parallel processing. Local routing and concurrent access feed into the enhancements to the Omega network provided by Ultra. First, the network is pipelined, which maximizes the use of local routing and allows a delay between messages that is equal to switch cycle time, not network transit time. This means that the network is message switched, and that switch settings are not maintained while awaiting reply. This strategy would normally have its own high blocking factor; to offset this, each switch has a queue which holds requests, so that the need for resubmission is reduced. And the destination and return adresses do not have to be transmitted with each message. Instead, the origin of a message entering the network is determined by its input port. This means that only the destination address is needed. By a simple algorithm, each stage of the network replaces the bit that sent the message to that stage with a bit replacement signifying the return address. When the message has reached its destination, the bit pattern that allowed the transmitting to the destination has been completely changed into the return address.

There are other issues associated with the network in Ultra, including the combination of requests and the implementation of the fetch-and-add primitive; they are reserved for discussion under control, in the next section of this paper.

6. ISSUES OF CONTROL

The possibilities of system operation in subsets under reconfiguration increases considerably the issues involved in control. First of all, control means here determining, maintaining and terminating the configuration itself, as well as (possibly) coordinating the subsets created. Routing of instruction streams is a central issue here, and particularly in MIMD mode becomes problematic, because each partition must have its own control structure. Much of what would under simple SISD processing be handled in hardware becomes in a reconfiguration environment a complex software issue. By looking at the issue of control in the four systems that were discussed in the previous section - the DC Group, the PASM architecture, the Star Local Network, and the NYU Ultracomputer - we will see some proposed solutions to the problems of control in sophisticated systems.

All of the issues involved in control cannot be discussed for all four systems, because the awareness of these issues varies from designer to designer. However, the systems under study do offer various and interesting solutions to the problems of control, and we will see that these solutions do not necessarily grow in complexity with the complexity of the overall systems, largely because there is a tradeoff between complexity and flexibility in larger systems.

6.1 The DC Group solution to the problem of control centers on two principles of reconfiguration in the system: 1) If there are n computer elements, CEs, consisting of processor and local memory, then there are potentially 1 to n number of possible subsets that can be formed, with from 1 to n possible different timing demands; 2) all of the possibly n different computers should be able to operate concurrently; and 3) the possible different combination of CEs is limited by the linear configuration of the system discussed in section 5 of this paper. Each CE must potentially have its own control unit, which must be coordinated with other units of other CEs in a computer that is constructed of more than 1 CE; that is, potentially n control units will have to function as one [Kar78].

Control issues and proposed solutions were described early by the Kartashevs for

their dynamic architecture [Kar78d] [Kar77]. Rejecting the synchronous and asychronous control organizations appropriate to systems with one central or several fixed local control units, they proposed a modular control organization. Originally thought of in the context of LSI technolology, each CE, synonymous with each LSI module, was provided with a local modular control device, MCD, which was capable of running a subset with a size of 1, but which was also capable of being coordinated with all other MCDs of a given configured subset up to size *n*.

The thinking here, originated in an earlier technology, has not changed, it seems, in its basic concepts. Each program instruction is written concurrently to all modules of a subset "computer," although it is unclear what overall control element of the system does this writing [Kar79a]. It is executed during one instruction cycle, but because the operand word size and memory speed vary, the MCD generates variable subcycles. But these subcycles are the same, of course, for all members of the subset computer. The MCD is the same for all elements in the subset, and processor dependent and data fetch intervals last the same time in all modules. The number of modules contained in a given subset does not affect sequencing or duration of instructions or cycles.

As we observed in section 5 of this paper, the DC group design allows mainly for linear communication between adjacent elements; thus, as Figure 6 shows, a system with 5 computer elements yields only 16 different configurations. This simplifies communication control somewhat, in that broadcasting of instructions among connected processors occurs by right- and left-transfer of the connecting bus. One can conceive of a subset, therefore, as that group of processors that has its outermost bus lines set in no-transfer mode. Transfer control, that is, the setting of the connecting bus into right-transfer, left-transfer, or no-transfer mode, is provided by a V monitor that is external to the group; if several units makes concurrent communication requests, the V monitor resolves conflicts on the basis of priority codes assigned to the programs being computed. The V monitor is also connected by a separate bus to every module. In a given subset, one module, the most

significant, transfers to the V monitor the control codes necessary for architectural transitions. Thus the V monitor is involved in both instruction requests and reconfiguration moments.

6.2 PASM control, unlike that of the Kartashev system, does not allow for the configuration of one processor element as one subset, and this limitation is evident in the control structure. The microcontrollers, MCs, are a set of microprocessors that act as control units for processors in SIMD mode and control the activities of the processors in MIMD mode [Sie81a]. If there are Q microcontrollers and N processors, then N/Q is the size of the smallest allowable partition. The number of allowable partitions is therefore equal to the number of microcontrollers. The PASM literature speaks normally of 1024 processors and 16 controllers, with a resulting 64 as the number of partitions.

Each MC is a unit consisting of a microprocessor and a memory element; like the processors themselves in PASM, the MCs have double memory elements so that memory loading and processing can go on simultaneously. When the subset is in SIMD mode, each MC fetches instructions from its memory element and executes control flow instructions, as well as broadcasting the data processing instructions to its connected processors. In MIMD mode the microcontrollers help coordinate the activities of their connected processors.

What seems to be unique to the PASM design is the notion of permanently assigning a given MC to a given subset of PEs. The other systems under study do not have this limitation. Because of this structure, the operating system only has to schedule and monitor the MCs; it never interfaces directly with the processors themselves. This suggests a special permanent subdividing of the overall system. The design also eliminates the need for a interconnection network allowing for communication among all processors, because a strong definition of precisely which processors need to talk to each other is determined from the outset. The obvious disadvantage of this system is that larger subsets can only grow by the order of two, and the total interconnection possibility of N! allowed by a full interconnection is not possible in PASM.

6.3 The Star network centers control of network routing in the switches. The switching element, which is modular and always constructed of a single type, is built of two major parts, called the control plane and the data plane [Wu82]. Data communication occurs in the data plane, and the control plane generates the control signals that establish connection paths used to transmit the data of the data plane.

The control plane sets up the path, by setting the switches, from the source to the destination, according to the routing scheme based on the modified network topology discussed in section 5. Through a set of input control lines, the control circuit receives signals from the previous stage in the network, develops control signals for its associated data plane, and sets up the signals for transmission to the next stage. The control plane has four internal registers to record the current connection status of the switching element.

Starnet is a circuit switching network, and with the above-described design the physical path for transmission is established in one clock period with two phases. In phase one, the request for connection is sent down the switches according to the routing scheme, and the control planes in each switch go through the handshaking process described above. If the request has been successful, and no conflict has been encountered, an acknowledge signal is generated by the receiver. This completes phase 1. During phase 2 the switching elements that have already been involved in the path establishing update their internal registers and set up the connection path. Thus, at the end of phase 2 the physical path is established; it will remain established as long as necessary, and until the source issues a signal to disconnect.

Within this scheme, during SIMD processing a controller broadcasts instructions to the processors that have been established as part of the subset for SIMD mode, and the instructions are then executed against the data stored in the associated memory. A task in SIMD mode is initiated when a task descriptor is sent by a cooperating processor to a VLSI processor unit that will serve as the controller. The task descriptor includes the number of processor units and the layout of the data streams. It is then the job of the controller to transmit the signals to connect the necessary processor units, and these individual

processor units establish necessary data paths to memory units.

In MIMD processing, when individual processors execute independently, the network capablity is used to establish configurations based on process needs; this is clearly one of the goals of a full interconnection network. The strategy in Star is called distributed scheduling; all free VLSI processors are equally accessable to a requesting controller, and no heirarchical or precedent relationship exists among the free processor units. When a task enters a cooperating processor, a task descriptor is formed to exploit parallel execution. The descriptor is passed by the cooperating processor to a free processor in a chain-of-command strategy to complete the parallel execution with as many processor connected as necessary. All connections in the communication network, connecting all cooperating processors, are maintained until completion of the task.

6.4 The NYU Ultracomputer makes use of a switch-oriented, local control scheme that is similiar to the one found in the Star network. However, while Star is circuit switched, the Ultracomputer is message switched. This means that full paths are not established from sender to receiver in a predictable cycle, and that switch settings are not held. Furthermore, the strategy of control is designed to maximize the goal of the system to provide for the kind of parallel processing described in section 5.

Control in the Ultracomputer involves maintaining the queue described in section 5, generation of destination and return addresses, and implementing of concurrent loads and stores [Got83a]. Because switch settings are not maintained, the system needs an elaborate method of keeping track of addresses. It does not transmit destination and return addresses with each message; rather it provides an elaborate algorithm that performs bit replacements at each stage of the network. Basically, the relevant bit that determined routing to a given switch is replaced, after use, as it were, with a bit that will allow for return. When the message has reached the destination, the destination address has been replaced, bit by bit, by the source address. Thus, storage for address in the message-switched packet is mimimized.

The most elaborate innovation in the Untracomputer is the strategy for combining requests to the same memory cell. Based on the serialization principle discussed in section 3, the following concurrent requests can be combined:

1) Load-Load : one of the requests is forwarded and the return is sent to each processor that generated the request;

2) Load-Store : The store is forwarded and the resulting value is returned to the processor requesting the load;

3) Store-Store : forward one store and discard the other.

These combinations can occur at any stage of the network. They can also be combined with the fetch-and-add operation at the switches, because the switches contain the necessary adder to implement the F&S. And a generalization of this design allows for a fetch-and-ϕ instruction, providing for other arithmetic functions. Thus, we can see that the special logical considerations of the Ultracomputer determine greatly issues of control in the interconnection network.

7. CONCLUSION

7.1 What Has Been Attempted in this Study. In order to discover where reconfiguration "comes from," and so that we could formulate some fundamental premises upon which to proceed with the analysis of later developments, our discussion began by looking at some early work in computation, both in theory and in the development of proposed designs. The early classic thinking on computation, the well known presentations of Turing and von Neumann, was examined first. We then looked at the efforts of Miller and Cocke to provide a theoretical framework for developing notions of reconfiguration, as well as the two early proposed systems of Estrin and associates and of Reddi and Feustel.

We saw that many of the motivations for reconfiguration appear early in the literature, but that the technology had not yet sufficiently developed to allow a fully developed set of motivations and criteria. We observed that reconfiguration appears very little in the early think on computing, because aspects of finite time and finite space do not influence that thinking. The early literature, therefore, does not provide us with a model for reconfiguration. Reconfiguration rises late, relatively speaking, in the development of the technology; it rises as a response to problems in the technology itself, rather than as a response to theory of algorithms and problem solving. Its model grows within the historical dimension of the development of the technology itself.

The focus of the discussion then turned to fault tolerance. The attempt was made to clarify the definition of fault tolerance and the issues involved in it, and to present a description and analysis of some of the major developments in architecture for fault tolerance. Siewiorek's conception of the stages of development in fault tolerant architectures has been regarded as a scheme that stops before the more advanced designs for fault tolerance. Only in the last two stages of his five-stage development toward "dynamic redundancy" can we begin to see what we call here reconfiguration. These last two stages were further discussed by an investigation of two specific systems, the Tandem computer and the C.vmp system, which represent the fourth and fifth stages of

Siewiorek's scheme. This discussion of dynamic redundancy was therefore followed by a discussion of some recent designs for reconfiguration, and "reconfiguration" is seen here as replacing "dynamic redundancy" when we begin to speak of the use of interconnection networks for fault tolerance. Investigation of the use of communication networks was demonstrated by three quite different designs, the Extra Stage Cube, the Gamma network, and the MPP system.

The goal of reconfiguration for fault tolerance is not the prevention of failure, but rather the manipulation of failure. The inherent tendency toward failure is countered by the potential for protection and recovery, mainly through the exploitation of another inherent tendency, the tendency toward permutations for protection. Because failure is a state that is planned for the design can be more adventurous. In early stages of fault tolerance, the tolerance is provided in all cases by redundancy, either time redundancy, usually provided by software, and basically characterized by repeated execution, or physical redundancy, most primitively characterized by the wheeling in of a new, duplicate system. However, while design can be less conservative, and while fault tolerance can become more accurate and efficient, the implementation of more advanced designs does not replace the basic process of redundancy; it simply makes this fundamental process more sophisticated. And its goal remains the same: the correct execution of a specified algorithm in the presence of defects.

The discussion then turned to reconfiguration for the sake of performance enhancement, largely for tasks in image processing and parallel processing. Many reconfigurable systems have been proposed, and the review considers the PM^4 system, the CHiP computer, and TRAC, as well as other proposals. This review demonstrated the range of issues involved in reconfiguration for performance enhancement, including the nature of the processors, the relationship of processors to memory, local memory versus global memory, scheduling and other issues of control, interconnection communication, and purpose for which the system is designed. Sections 5 and 6 of the report discussed interconnections strategies and control in four other proposed systems, which were

deemed to be the most fully developed in the literature: the dynamic architecture of the Kartashevs, the PASM architecture, the Star local network, and the NYU Ultracomputer.

In the remainder of this conclusion, some observations on the tendencies in the design of a reconfigurable architecture will be attempted, and some remarks will be made on further areas of research that would extend our understanding of the subject.

7.2 The Nature of Reconfiguration. When a system undergoes reconfiguration, its nature as a whole is changed because of the demands of a specific task, and this change may result in the partitioning of the system, and therefore the creation of subsystems. Advances in research in VLSI technology have made it feasible to consider the implementation of massive and complex parallel architectures built of thousands of processors, which provide enormous throughput; this potential alters radically the notion of what constitutes the set of computable problems. But the availability of such massive power is not alone the solution to all computation. These large numbers of processors can be configured in different ways, to perform SIMD- and MIMD-based tasks, among others. It is clear that not only masses of processors, but also their configuration, lead to efficient complexity. This leads to the problem of the degree of match between algorithm and architecture that efficient complexity implies. A system with a fixed architecture will only match a small set of the computationally complex algorithms that exist. It is well known that a massively parallel system, when mismatched with a task demanding a different configuration, experiences performance degradation. Thus we have the justification for our interest in the development of architectures that can reconfigure into a different complexity, under the control of software. The goal here is proper match between algorithms and architectures, no matter what the complexity and demands of the algorithms.

An important issue in designing a reconfigurable architecture is the nature of communication in the system, both among the elements in a subset and among the subsets of the entire system. Complexity in algorithms often means complexity in communication needs among processors, memory, and I/O devices. Reconfiguration in multiprocessing

environments places extra demands on communication, and this topic often dominates serious investigation. Communication figures in control of partitioning, scheduling, and other system issues, as well as in processor and memory communication in the reconfigured subset. Goals for communication are *total* communication - the highest possible number of linkage permutations among modules - but also the least possible complexity and cost. A popular approach to communication is the implementation of the multistage interconnection network. In spite of its delay, its relatively high blocking factor, and the implications of its inevitable crossover lines in a VLSI environment [Fra81], the multistage ICN remains attractive because of its limited growth rate when large numbers of connected elements are being considered. In the light of the discussion of this issue in the literature, some strategies for interconnection can be seen to be much too limited, most obviously the linear bus connection strategy of the Kartashev dynamic architecture. And the multistage interconnection network seems to work; recent reports on the Star system [Wu85], PASM [Dav85], and TRAC [Des85] all report favorably on its use. This is especially true of the implementation of the TRAC prototype in which the use of banyan interconnection network is considered to be the most important contribution of the TRAC project.

Two important aspects of the implementation of a multistage interconnection network should also be mentioned here. The first is the problem of local *versus* global memory, which results, when dealing with an interconnection network, in the issue of whether to attach local memory to given processors or to have global memory that is accessed by all processors via the ICN. By its very nature the multiprocessing environment is meant to obliterate the "von Neumann bottleneck," the problem of one processor at the center of a powerful system; but the design strategy that demands access to memory by processors over an ICN runs the risk of creating a new bottleneck, here not in the processing, but rather in the communication link. As we have seen local memory seems a solution here, but sophisticated use of the interconnection network, particularly the

strategies employed in the NYU Ultracomputer, is a solution that allows use of global memory. The other aspect of ICN implementation is the nature of communication beyond mere topology, specifically the methods employed in setting switches. Early plans for external control of switches seem to have given way to methods of local switch control, which decrease blocking and allow greater flexibility. The use in PASM of the extra stage cube topology is representative here. But also of concern is the issue of whether or not the network should be circuit switched, message switched, or both. The Ultracomputer, with its queueing at switches and its combining of instructions at switches, represents a sophisticated approach to message switching in an interconnection network. The PASM cube allows both circuit and message switching, and also of interest is the TRAC system, whose banyan network is capable of implementing both circuit and message switching.

Much of this discussion does indeed focus on multistage interconnection networks for both fault tolerance and performance enhancement; however, it would be narrow in focus to think of the communications needs of reconfigurable architecture in these terms. We have seen, for example, the lattice structures employed in the CHiP architecture, and the importance of the 4N grid communication strategy employed by the MPP system.

The development of interconnection strategies dominates reconfiguration for both fault tolerance and performance. This suggests a close affinity between these two design issues. Advances in communication and control can be employed for either purpose. However, our analysis seems to indicate that the connections between fault tolerance and performance must be carefully limited. Redundancy is an important dividing point: redundancy is the center of reconfiguration for fault tolerance, whereas maximization of resources, with a minimum of overlap of redundancy of resources, is the purpose of performance enhancement.

7.3 Suggestions for Further Study. As stated above, this survey begins at the advent of VLSI technology, but we observe in the systems under study a need for stronger impact of the new technology on the thinking about system design. Certainly

multiprocessing systems will make use of chip advances for the individual processors in the system; but we have seen a desire to use off-the-shelf processors, rather than attempts at individual design; and most obvious is the persistence of the attraction of communication links that are not chip-based, and which have as their performance criteria pre-VLSI considerations, mainly the problem of growth in the number of switches in a network. Fault-tolerant circuit layout designs, including spare row and column organization, enhance integrated circuit yield [Moo86]. There are of course problems of cost and chip-pin ratios with the technology. This is a complex issue and demands consideration that would expand greatly the scope of the present study.

One of the most interesting aspects of reconfiguration is the pre-analysis of algorithms, and the growing investigation of the union of actual processes with architecture. The high-level language program is a view of one single system carrying out a sequence of computations; on the level of the machine, a different view prevails, one in which the execution of instructions, allocation of resources, and structure of commumications is many-layered and representative of the actual process in a different way. Many of the systems under study are structured for the task environment. The CHiP system, for example, in an obvious way shows reconfiguration of its lattice network for the sake of process. The tendency here is beyond reconfiguration for the sake of creating a general-purpose machine, to reconfiguration to the sake of specific purposes in a specific environment. It was stated in the beginning of this paper that reconfiguration perhaps stands in opposition to the tendency toward dedicated systems; but with the potential of reconfiguration within specific task environments, most notably image processing, we see the development of an interest in reconfiguration that does not make a machine general purpose, but rather oriented to a predefined subset of tasks. A report on the PASM project, for example, deals with the uniting of the design of the system with the specific task of contour analysis for image processing [Tuo83].

The recent efforts toward designing a reconfigurable architecture are emerging

beyond the stage of paperwork design into the stage of implementation. The recent report on TRAC announces an up-and-running prototype, with a developed instruction set and operating system. The originally proposed Banyan network has been successfully implemented. Packet switching allows asychronous communication among the TRAC processors, and the network supports the dynamic generation of the three tree-shaped, circuit-switched communication structures - shared tree, data tree, and instruction tree - that were in the original design [Des85]. Also of interest is the development at IBM of the Research Parallel Processor Prototype (RP3), which will attempt to implement the research efforts of both the NYU Ultracomputer and the Caltech Cosmic Cube in a full-scale research-oriented machine supporting 512 microprocessors [Pfi85]. It is reported that performance evaluation and detailed physical and logical design have already provided results, and that the machine will be kept as an open project, allowing collaboration with other organizations.

8. BIBLIOGRAPHY

[Ada82] G.B. Adams III and H.J. Siegel, "The extra stage cube: A fault-tolerant interconnection network for supersystems," *IEEE Trans. Comput.*, vol. C-31, pp. 443-454, May 1982.

[Agr83] D.P. Agrawal, "Graph theoretical analysis and design of multistage interconnection networks," *IEEE Trans. Comput.*, vol. C-32, pp. 637-648, July 1983.

[Agr82] D.P. Agrawal, "Testing and fault-tolerance of multistage interconnection networks," *Computer*, vol. 15, pp. 41-53, Apr. 1982.

[Ale86] N.A. Alexandridis, "Adaptable software and hardware: Problems and solutions," *Computer*, vol. 19, pp. 29-39, Feb. 1986.

[Arv80] Arvind, "Decomposing a program for multiple processor systems," in *Proc. 1980 Int. Conf. Parallel Processing*, 1980, pp. 7-14.

[Bar68] G. Barnes, R. Brown, M. Kato, D. Kuck, D. Slotnick, and R. Stokes, "The Illiac IV computer," *IEEE Trans. Comput.*, vol. C-17, pp. 746-757, Aug. 1968.

[Bar78] J.F. Bartlett, "A 'NonStop' operating system," in *Proc. 11th Int. Conf. on Syst. Sciences*, Hawaii, 1978, pp. 103-119.

[Bat80] K.E. Batcher, "Design of a massively parallel processor," *IEEE Trans. Comput.*, vol. C-19, pp. 836-840, Sept. 1980.

[Bri79] F.A. Briggs, F.-S. Fu, K. Hwang, and J.H. Patel, "PM^4 - a reconfigurable multiprocessor system for pattern recognition and image processing," in *Proc. 1979 Nat. Comput. Conf.*, AFIPS, vol. 48, 1979, pp. 255-265.

[Bro83] G. Broomell and J.R. Heath, "Classification categories and historical development of circuit switching topologies," *Computing Surveys*, vol. 15, pp. 95-133, June 1983.

[Bur46] A.W. Burks, H.H. Goldstein, and J. von Neumann, "Preliminary discussion of the logical design of an electronic computing instrument," Part I, Vol.1, *Report prepared for U.S. Army Ord. Dept.*, 1946; reprinted in J. von Neumann, *Collected Works*, Vol. V, Pergamon Press, 1963.

[Che84] S. Chen and G.X. Ritter, "Reconfigurable architecture for image processing," in *Proc. Int. Conf. on Computer Design*, 1984, pp. 516-519.

[Cla82] E.M. Clarke and C.N. Nikolaou, "Distributed reconfiguration strategies for fault-tolerant multiprocessor systems," *IEEE Trans. Comput.*, vol. C-31, pp. 771-784, Aug. 1982.

[Cli85] C.L. Cline and H.J. Siegel, "Augmenting Ada for SIMD parallel processing," *IEEE Trans. Soft. Eng.*, vol. SE-11, pp. 970-977, Sept. 1985.

[Dav85a] N.J. Davis IV and H.J. Siegel, "The PASM prototype interconnection network design," in *Proc. 1985 Nat. Comput. Conf.*, AFIPS, vol. 54, 1982, pp. 183-190.

[Dav85b] N.J. Davis III, W. T.-y Hsu, and H.J. Siegel, "Fault location techniques for distributed control interconnection networks," *IEEE Trans. Comput.*, vol. C-34, pp. 902-910, Oct. 1985.

[Dav82] C.G. Davis, S.P. Kartashev, and S.I. Kartashev, "Reconfigurable multicomputer networks for very fast real-time applications," in *Proc. 1982 Nat. Comput. Conf.*, AFIPS, vol. 51, 1982, pp. 167-184.

[Den70] P.J. Denning, "Virtual Memory," *Computing Surveys*, vol. 2, pp. 153-189, Sept. 1970.

[Des85] S.J. Deshpande, R.M. Jenevein, and G.J. Lopovski, "TRAC: An experience with a novel architectural prototype," in *Proc. 1985 Nat. Comput. Conf.*, AFIPS, vol. 54, 1982, pp. 247-258.

[Dig76] Digital Equipment Company, "Pdp-11 04/34/45/55 processor handbook," 1976.

[Edl82] J. Edler, A. Gottlieb, C.P. Kruskal, K.P. McAuliffe, L. Rudolph, M. Snir, P.J. Teller, and J. Wilson, "Issues related to MIMD shared-memory computers: The NYU Ultracomputer approach," in *Proc. 9th Annual Symp. Comput. Arch.*, Apr. 1982, pp. 63-72.

[Esf85] A.-H. Esfahanian and S.L. Hakimi, "Fault-tolerant routing in DeBruijn communication networks," *IEEE Trans. Comput.*, vol. C-34, pp. 777-788, Sept. 1985.

[Est60] G. Estrin, "Organization of computer systems: The fixed plus variable structure computer," in *Proc. Western Joint Computer Conf.*, 1960, pp. 33-40.

[Est63a] G. Estrin, B. Russell, R. Turn, and J. Bibb, "Parallel processing in a restructurable computer system," *IEEE Trans. Comput.*, vol. EC-12, pp. 747-755, Dec. 1963.

[Est63b] G. Estrin and R. Turn, "Automatic assignment of computations in a variable structure computer system," *IEEE Trans. Comput.*, vol. EC-12, pp. 755-773, Dec. 1963.

[For85] J.A.B. Fortes and C.S. Raghavendra, "Gracefully degradable processor arrays," *IEEE Trans. Comput.*, vol. C-34, pp. 1033-1044, Nov. 1985.

[Fot61] J. Fotheringham, "Dynamic storage allocation in the Atlas computer including an automatic use of a backing store," *Comm. ACM*, vol. 4, pp. 435-436, Oct. 1961.

[Fra81] M.A. Franklin, "VLSI performance comparison of banyan and crossbar communications networks," *IEEE Trans. Comput.*, vol. C-30, pp. 283-291, Apr. 1981.

[Feu73] E.A. Feustel, "On the advantages of tagged architecture," *IEEE Trans. Comput.*, vol. C-22, pp. 644-656, July 193.

[Gan81] D.B. Gannon and L. Snyder, "Linear recurrence algorithms for VLSI: The configurable, highly parallel approach," in *Proc. 1981 Int. Conf. Parallel Processing*, 1981, pp. 259-260.

[Gaj85] D.D. Gajski and J.-K. Peir, "Essential issues in multiprocessor systems," *Computer*, vol. 18, pp. 9-27, June 1985.

[Got83a] A. Gottlieb, R. Grishman, C.P. Kruskal, K.P. McAuliffe, L. Rudolph, and M. Snir, "The NYU Ultracomputer - Designing an MIMD shared memory parallel computer," *IEEE Trans. Comput.*, vol. C-32, pp. 175-189, Feb. 1983.

[Got83b] A. Gottlieb, B. D. Lubachevsky, and L. Rudolph, "Basic techniques for the efficient coordination of very large numbers of cooperating sequential processors," *ACM TOPLAS*, Jan. 1983, pp. 164-189.

[Gra82] F.G. Gray, "General purpose reconfigurable architecture," in *Proc. Int. Conf. on Circuits and Computers*, 1982, pp. 122-125.

[Hwa84] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing*, McGraw-Hill Book Co., 1984.

[Iwa85] K. Iwasaki, N. Yamaguchi, T. Shimura, Y. Hagawara, T. Funabashi, and K. Minorikawa, "Design methodology for reconfigurable module-structured VLSI," in *Proc. 1985 Custom Integrated Circuits Conf.*, 1985, pp. 464-467.

[Jen82] R.M. Jenevein and J.C. Browne, "A control processor for a reconfigurable array computer," in *Proc. 9th Annual Symp. Comput. Arch.*, Apr. 1982, pp. 81-89.

[Kap80] R.N. Kapur, "Organization of the TRAC processor-memory subsystem," in *Proc. 1980 Nat. Comput. Conf.*, AFIPS, vol. 49, 1982, pp. 623-629.

[Kar86a] S.P. Kartashev and S.I. Kartashev, "Data exchange optimization in reconfigurable binary trees," *IEEE Trans. Comput.*, vol. C-35, pp. 257-273, Mar. 1986.

[Kar86b] ——, "Adaptable software for dynamic architectures," *Computer*, vol. 19, pp. 61-77, Feb. 1986.

[Kar83] ——, "Reconfigurable fault-tolerant multicomputer network," in *Proc. 1983 Nat. Comput. Conf.*, AFIPS, vol. 52, 1983, pp. 595-610.

[Kar82a] ——, "Distribution of programs for a system with dynamic architecture," *IEEE Trans. Comput.*, vol. C-31, pp. 488-514, June 1982.

[Kar82b] ——, "Designing and programming supersystems with dynamic architectures," in *Designing and Programming Modern Computer Systems*, Vol. 1, Chapter III, *LSI Modular Computer Systems*, eds. S.P. Kartashev and S.I. Kartashev, Prentice-Hall, Inc., 1982, pp. 245-385.

[Kar81] ——, "Reconfiguration of dynamic architecture into multicomputer networks," in *Proc. 1981 Int. Conf. on Parallel Processing*, 1981, pp. 1233-140.

[Kar80a] ——, "Problems of designing supersystems with dynamic architectures," *IEEE Trans. Comput.*, vol. C-29, pp. 1114-1131, Dec. 1980.

[Kar80b] ——, "Architectures for supersystems of the '80s," in *Proc. 1980 Comput. Conf.*, AFIPS, vol. 49, 1980, pp. 165-180.

[Kar79a] ——, "A multicomputer system with dynamic architecture," *IEEE Trans. Comput.*, vol. C-28, pp. 704-721, Oct. 1979.

[Kar79b] ——, "Performance of reconfigurable busses for dynamic architectures," in *Proc. 1st. Int. Conf. on Distributed Comput. Syst.*, Huntsville, AL, 1979, pp. 261-273.

[Kar79c] ——, and C. V. Ramamoorthy, "Adaptation properties for dynamic architectures," in *Proc. 1979 Nat. Comput. Conf.*, AFIPS, vol. 48, 1979, pp. 543-556.

[Kar78a] ——, "Dynamic architectures: Problems and solutions," *Computer*, vol. 11, pp. 26-40, July 1978.

[Kar78b] ——, "LSI modular computers, systems and networks," *Computer*, vol. 11, pp. 7-10, July 1978.

[Kar78c] ——, "Software problems for dynamic architectures," in *Proc. 2nd Int. Comput. Software Appl. Conf.*, Chicago, IL, 1978, pp. 775-780.

[Kar78d] ——, "Selection of the control organization for a multicomputer system with dynamic architecture," in *Proc. 4th Euromicro Symp. on Microprocessing and Microprogramming*, Munich, 1978.

[Kar77] ——, "A microprocessor with modular control as a universal building block for complex computers," in *Proc. 3rd Euromicro Symp. on Microprocessing and Microprogramming*, Amsterdam, 1977, pp. 210-216.

[Kat78] D. Katsuki, E.S. Elsam, W.F. Mann, E.S. Roberts, J.G. Robinson, F.S. Skowronski, and E.W. Wolf, "Pluribus - An operational fault-tolerant multiprocessor," *Proc. IEEE*, vol. 66, Oct. 1978, pp. 1146-1159.

[Kat78a] J.A. Katzman, "A fault-tolerant computing system," in *Proc. 11th Int. Conf. on Syst. Sciences*, Hawaii, 1978, pp. 85-102.

[Kru82] C.P. Kruskal, "Algorithms for replace-add based paracomputers," in *Proc. 1982 Int. Conf. Parallel Processing*, 1982, pp. 219-223.

[Kue85] J.T. Kuehn and H.J. Siegel, "Extensions to the C programming language for SIMD/MIMD parallelism," in *Proc. 1985 Int. Conf. Parallel Processing*, 1985, pp. 232-235.

[Kue85b] J.T. Kuehn, H.J. Siegel, D.L. Tuomenoksa, and G.B. Adams III, "The Use and Design of PASM," in *Integrated Technology for Parallel Image Processing*, ed. S. Levialdi, Academic Press, London, 1985.

[Lal85] P.K. Lala, *Fault Tolerant & Fault Testable Hardware Design*. Englewood Cliffs, NJ: Prentice-Hall, 1985.

[Lin84] W. Lin and C.-L. Wu, "Object-based resource management mechanism for a high-performance distributed computing system - Star," in *Proc. Compsac 84*, Nov. 1984, pp. 208-216.

[Lip77] G.J. Lipovski and A. Tripathi, "A reconfigurable varistructured array processor," in *Proc. Int. Conf. on Parallel Processing*, 1977, pp. 165-174.

[Lun80] S.F. Lundstrom and G.H. Barnes, "A controllable MIMD architecture," in *Proc. 1980 Int. Conf. Parallel Processing*, 1980, pp. 19-27.

[Ma82] Y.W. Ma, "Reconfiguration control algorithms for reconfigurable computer systems," in *Proc. 6th Int. Comput. Software Appl. Conf.*, Chicago, IL, 1982, pp. 70-77.

[McM82a] R.J. McMillen and H.J. Siegel, "Routing schemes for the augmented data manipulator network in an MIMD system," *IEEE Trans. Comput.*, vol. C-31, pp. 1202-1214, Dec. 1982.

[McM82b] R.J. McMillen and H.J. Siegel, "Performance and fault-tolerance in the augmented data manipulator network," in *Proc. 9th Annual Symp. Comput. Arch.*, Apr. 1982, pp. 63-72.

[Mar83] E.W. Martin, "Strategy for a DoD software initiative," *Computer*, vol. 16, pp. 52-59, Mar. 1983.

[Mas79] G.M. Masson, G.C. Gingher, and S. Nakamura, "A sampler of circuit switching networks," *Computer*, vol. 12, pp. 32-48, June 1979.

[Mey85] D.G. Meyer, H.J. Siegel, T. Schwederski, N.J. Davis, and J.T. Kuehn, "The PASM parallel system prototype," in *Proc. Compcon*, Feb. 1985, pp. 429-434.

[Mil74] R.E. Miller and J. Cocke, "Configurable computers: A new class of general purpose machines," in *Int'l Sym. Theoretical Programming*, ed. A. Ershov and V.A. Nepomniaschy (Lecture Notes in Computer Science, ed. G. Goos and J. Hartmanis, vol. 5), Springer-Verlag, 1975, pp. 319-337.

[Moo86] W.R. Moore, "A review of fault-tolerant techniques for the enhancement of integrated circuit yield," *Proc. IEEE*, vol. 74, pp. 684-698, May 1986.

[Ng80] Y.W. Ng and A. Avizienis, "A unified reliability model for fault-tolerant computers," *IEEE Trans. Comput.*, vol. C-29, pp. 1002-1011, Nov. 1980.

[Orn75] S.M. Ornstein, W.R. Crowther, M.F. Kraley, R.D. Bressler, A. Michel, and F.E. Heart, "Pluribus - A reliable multiprocessor," in *Proc. 1975 Nat. Comput. Conf.*, AFIPS, vol. 44, 1975, pp. 551-559.

[Par84] D.S. Parker and C.S. Raghavendra, "The Gamma network," *IEEE Trans. Comput.*, vol. C-33, pp. 367-373, Apr. 1984.

[Par82] D.S. Parker and C.S. Raghavendra, "The Gamma network: A multiprocessor interconnection network with redundant paths," in *Proc. 9th Annual Symp. Comput. Arch.*, Apr. 1982, pp. 73-80.

[Pea77] M.C. Pease III, "Indirect binary *n*-cube microprocessor array," *IEEE Trans. Comput.*, vol. C-26, pp. 458-473, May, 1977.

[Pfi85] G.F. Pfister, W.C. Brantley, D.A. George, S.L. Harvey, W.J. Kleinfelder, K.P. McAuliffe, E.A. Melton, V.A. Norton, J. Weiss, "IBM Research Parallel Processor Prototype (RP3) : Introduction and architecture," in *Proc. 1985 Int. Conf. Parallel Processing*, 1985, pp. 764-771.

[Pra85a] D.K. Pradhan, "Fault-tolerant multiprocessor link and bus network architectures," *IEEE Trans. Comput.*, vol. C-34, pp. 33-45, Jan. 1985.

[Pra85b] D.K. Pradhan, "Dynamically restructurable fault-tolerant processor network architectures," *IEEE Trans. Comput.*, vol. C-34, pp. 434-447, May 1985.

[Pre80] U.V. Premkumar, R. Kapur, M. Malek, G.J. Lipovski, and P. Horne, "Design and implementation of the banyan interconnection network in TRAC," in *Proc. 1980 Nat. Comput. Conf.*, AFIPS, vol. 49, 1982, pp. 643-653.

[Rag86] C.S. Raghavendra and A. Varma, "Fault-tolerant multiprocessors with redundant-path interconnection networks," *IEEE Trans. Comput.*, vol. C-35, pp. 307-316, Apr. 1986.

[Ram86] C.V. Ramamoorthy and Y.W.E. Ma, "Optimal reconfiguration strategies for reconfigurable computer systems with no repair," *IEEE Trans. Comput.*, vol. C-35, pp. 278-280, Mar. 1986.

[Red75] S.S. Reddi and E.A. Feustel, "An approach to restructurable computer systems," in *Parallel Processing*, ed. T.-y. Feng (Lecture Notes in Computer Science, ed. G. Goos and J. Hartmanis, vol. 24), Springer-Verlag, 1975, pp. 319-337.

[Red78] S.S. Reddi and E.A. Feustel, "A restructurable computer system," *IEEE Trans. Comput.*, vol. C-27, pp. 1-20, Jan. 1978.

[Red84] S.M. Reddy and V.P. Kumar, "On fault-tolerant multistage interconnection networks," in *Proc. 1984 Int. Conf. Parallel Processing*, 1984, pp. 155-164.

[Ree80] A.P. Reeves, J.D. Bruner, and M.S. Poret, "The parallel language Parallel Pascal," in *Proc. 1980 Int. Conf. Parallel Processing*, 1980, pp. 5-6.

[Ren84] D.A. Rennels, "Fault-tolerant computing - Concepts and examples," *IEEE Trans. Comput.*, vol. C-33, pp. 1116-1129, Dec. 1984.

[Sam86] M. Sami and R. Stefanelli, "Reconfigurable architectures for VLSI processing arrays," *Proc. IEEE*, vol. 74, pp. 712-722, May 1986.

[Sam83] M. Sami and R. Stefanelli, "Reconfigurable architectures for VLSI processing arrays," in *Proc. 1983 Nat. Comput. Conf.*, AFIPS, vol. 52, 1983, pp. 565-577.

[Sch86] T. Schwederski and H.J. Siegel, "Adaptable software for supercomputers," *Computer*, vol. 19, pp. 40-48, Feb. 1986.

[Sei85] C.L. Seitz, "The Cosmic Cube," *Comm. ACM*, vol. 28, pp. 22-33, Jan. 1985.

[Sej80] M.C. Sejnowski, E.T. Upchurch, R.N. Kapur, D.P.S. Charlu, and G.J. Lipovski, "An overview of the Texas Reconfigurable Array Computer," in *Proc. 1980 Nat. Comput. Conf.*, AFIPS, vol. 49, 1980, pp. 631-641.

[She84] J.P. Shen and J.P. Hayes, "Fault-tolerance of dynamic-full-access interconnection networks," *IEEE Trans. Comput.*, vol. C-33, pp. 241-248, Mar. 1984.

[Sie81a] H.J. Siegel, L.J. Siegel, F.E. Kemmerer, P.T. Mueller, Jr., H.E. Smalley, Jr., and S.D. Smith, "PASM: A partitionable SIMD/MIMD system for image processing and pattern recognition," *IEEE Trans. Comput.*, vol. C-30, pp. 934-947, Dec. 1981.

[Sie81b] H.J. Siegel and R.J. McMillen, "Using the augmented data manipulator in PASM," *Computer*, vol. 14, pp. 25-34, 1981.

[Sie80] H.J.Siegel, "The theory underlying the partitioning of permutation networks," *IEEE Trans. Comput.*, vol. C-29, pp. 791-800, Sept. 1980.

[Sie79a] H.J. Siegel, "Interconnection networks for SIMD machines," *Computer*, vol. 12, pp. 57-65, June, 1979.

[Sie79b] H.J. Siegel, R.J. McMillen, and P.T. Mueller, Jr., "A survey of interconnection methods for reconfigurable processing systems," in *Proc. 1979 Nat. Comput. Conf.*, AFIPS, vol. 48, 1979, pp. 529-542.

[Sie78a] H.J. Siegel, P.T. Mueller, Jr., and H.E. Smalley, Jr. "Control of a partitionable multimicroprocessor system," in *Proc. 1978 Int. Conf. on Parallel Processing*, 1978, pp. 9-17.

[Sie78ab H.J. Siegel and S.D. Smith, "Study of multistage SIMD interconnection networks," in *Proc. 5th Symp. Comput. Arch.*, Apr. 1978, pp. 223-229.

[Sie84] D.P. Siewiorek, "Architecture of fault-tolerant computers," *Computer*, vol. 17, pp. 9-18, Aug. 1984.

[Sie82] D.P. Siewiorek and R.S. Swartz, *The Theory and Practice of Reliable System Design*, Digital Press, 1982.

[Sie78] D.P. Siewiorek, V. Kini, H. Mashburn, S. McConnel, and M. Tsao, "A case study of C.mmp, Cm*, and C.vmp: Part 1 - Experiences with fault tolerance in multiprocessor systems," *Proc. IEEE*, vol. 66, Oct. 1978, pp. 1178-1199.

[Sie77] D.P. Siewiorek, M. Canepa, and S. Clark, "C.vmp: The architecture of a fault-tolerant multiprocessor," in *Proc. 1977 Int. Symp. Fault-tolerant Computing*, June, 1977.

[Sny82] L. Snyder, "Introduction to the configurable, highly parallel computer," *Computer*, vol. 15, pp. 47-56, Jan. 1982.

[Tru82] V.A. Trujillo, "System architecture of a reconfigurable multimicroprocessor research system," in *Proc. 1982 Int. Conf. on Parallel Processing,* 1982, pp. 350-352.

[Tuo84] D.L. Tuomenoksa and H.J. Siegel, "Task preloading for reconfigurable parallel processing systems," *IEEE Trans. Comput.,* vol. C-33, pp. 895-905, Oct. 1984.

[Tuo83] D.L. Tuomenoksa, G.B. Adams III, H.J. Siegel, and O.R. Mitchell, "A parallel algorithm for contour extraction: Advantages and architectural implications," in *Proc. 1983 Comp. Soc. Symp. on Computer Vision and Pattern Recognition,* 1983, pp. 336-344.

[Tur36] A.M. Turing, "On computable numbers, with an application to the Entscheidungsproblem," *Proc. London Math. Soc.*, ser. 2, vol. 42, pp. 230-265, 1936-37; corrections, *ibid.*, vol. 43, pp. 544-546, 1937; reprinted in *The Undecidable*, ed. M. Davis, Raven Press, 1965, pp. 115-154.

[Vic80] C. R. Vick, S. P. Kartashev, and S. I. Kartashev, "Adaptable architectures for supersystems," *Computer,* vol. 13, pp. 17-35, Nov. 1980.

[Yal85] S. Yalamanchili and J.K. Aggarwal, "Reconfiguration strategies for parallel architectures," *Computer,* vol. 18, pp. 44-61, Dec. 85.

[Wu85] C.-L. Wu, M. Lee, C. Sudtikitpisan, J. Moaddeb, G. Brown, W. Lin, N. Bagherzadeh, and D. Vaughn, "Prototype of Star architecture - a status report," in *Proc. 1985 Nat. Comput. Conf.,* AFIPS, vol. 54, 1982, pp. 191-201.

[Wu82] C.-L. Wu, T.-y. Feng, M.-C. Lin, "Star: A local network system for real-time management of imagery data," *IEEE Trans. Comput.,* vol. C-31, pp. 923-933, Oct. 1982.

[Wu80] C.-L. Wu and T.-y. Feng, "On a class of multistage interconnection networks," *IEEE Trans. Comput.,* vol. C-29, pp. 694-702, Aug. 1980.

[Wu78] C.-L. Wu and T.-y. Feng, "Routing techniques for a class of multistage interconnection networks," in *Proc. 1978 Int. Conf. on Parallel Processing,* 1978.

[Zim80] H. Zimmerman, "OSI reference model - The ISO model of architecture for open systems interconnections," *IEEE Trans. Commun.,* vol. COM-28, pp. 425-432, Apr. 1980.