

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

ARO 18790.17-EL-
APP-C
②

FAULT TOLERANT INTERCONNECTION NETWORKS AND IMAGE PROCESSING APPLICATIONS FOR THE PASM PARALLEL PROCESSING SYSTEMS

AD-A167 621

Ph.D. Thesis by:
George B. Adams III

Faculty Advisor:
Howard Jay Siegel

Appendix C for
**Distributed Computing for
Signal Processing:
Modeling of Asynchronous
Parallel Computation
Final Report**

U.S. Army Research Office
Contract No. DAAG29-82-K-0101*

S DTIC
ELECTE **D**
APR 30 1986
E

DTIC FILE COPY

*Chapters 1, 2, 5, and 9 supported by this contract.

This document has been approved
for public release and sales its
distribution is unlimited.

86 4 28 182

**A FAULT-TOLERANT INTERCONNECTION NETWORK
AND IMAGE PROCESSING APPLICATIONS FOR
THE PASM PARALLEL PROCESSING SYSTEM**

A Thesis
Submitted to the Faculty
of
Purdue University

by
George Bunch Adams III

In Partial Fulfillment of the
Requirements for the Degree
of
Doctor of Philosophy

December 1984



Registration For	
PH.D.	<input checked="" type="checkbox"/>
M.S.	<input type="checkbox"/>
B.S.	<input type="checkbox"/>
Other	
Department	
College	
Advisor	
Committee	

A-1

This document has been approved
for public release and is being
distributed in unlimited quantities.

to my parents

ACKNOWLEDGEMENTS

I am deeply grateful to Professor H. J. Siegel for his guidance and for creating an environment supporting and encouraging professional growth. I wish to thank the following people for their comments on this work: Professor D. B. Gannon, Professor L. H. Jamieson, Professor B. W. Wah, Professor P. H. Swain, Dr. R. J. McMillen, Dr. D. L. Tuomenoksa, J. T. Kuehn, and Dr. T. A. Grogan. Thanks to S. E. Katz for preparing the illustrations, and thanks to W. L. Booth, N. E. Lein, P. A. Loomis, M. J. Krebs, V. J. Spence, and L. L. Stovall for assistance in typesetting the manuscript. Thanks to E. N. Tinoco for the aircraft design information in Chapter 1. My thanks to the Research Institute for Advanced Computer Science for the use of its facilities in the completion of this document.

This work was supported by the Air Force Office of Scientific Research, Air Force Systems Command, USAF, under Grant AFOSR-78-3581; the Ballistic Missile Defense Agency under Grant DASG60-80-C-0022; the National Science Foundation under Grant ECS 80-16580; the Defense Mapping Agency, monitored by the United States Air Force Command, Rome Air Development Center, under Contract F30602-81-C-0193; the United States Army Research Office, Department of the Army, under Grant DAAG29-82-K-0101; and Rome Air Development Center under Contract F30602-83-K-0119.

TABLE OF CONTENTS

	Page
LIST OF TABLES	viii
LIST OF FIGURES	x
ABSTRACT	xviii
CHAPTER 1 INTRODUCTION	1
1.1 Motivation	1
1.2 Overview	8
1.3 Parallel Computer System Terminology	10
1.4 Interconnection Network Terminology	16
1.5 Summary	21
CHAPTER 2 SURVEY OF SELECTED PARALLEL COMPUTER SYSTEMS	22
2.1 Introduction	22
2.2 PASM	23
2.3 Numerical Aerodynamic Simulation Facility	34
2.4 Texas Reconfigurable Array Computer	43
2.5 STARAN	46
2.6 PUMPS	51
2.7 Conclusions	54
CHAPTER 3 DEFINITION OF THE EXTRA STAGE CUBE INTERCONNECTION NETWORK	55
3.1 Introduction	55
3.2 Definition of the Generalized Cube Network	56
3.3 Properties of the Generalized Cube Network	61
3.4 Definition of the Extra Stage Cube Network	62
3.5 Conclusions	69

	Page
CHAPTER 4 PROPERTIES OF THE EXTRA STAGE CUBE INTERCONNECTION NETWORK	70
4.1 Introduction	70
4.2 Fault-Tolerance Model	71
4.3 Single-Fault Tolerance	73
4.3.1 One-to-One Connections	73
4.3.2 Broadcast Connections	77
4.3.3 Finding Fault-Free Paths	82
4.4 Multiple-Fault Tolerance	85
4.5 Routing Tags	97
4.6 Partitioning	106
4.7 Permuting	113
4.8 Conclusions	118
 CHAPTER 5 SURVEY OF FAULT-TOLERANT MULTISTAGE INTERCONNECTION NETWORKS AND COMPARISON TO THE EXTRA STAGE CUBE	 122
5.1 Introduction	122
5.2 Survey	124
5.2.1 Modified Baseline Network	125
5.2.2 Augmented Delta Network	127
5.2.3 Multipath Omega Network	130
5.2.4 F-Network	136
5.2.5 Enhanced Inverse Augmented Data Manipulator Network	139
5.2.6 Gamma Network	145
5.2.7 Fault-Tolerant Beneš Network	149
5.2.8 Augmented C-Network	154
5.2.9 β -Networks	158
5.3 Comparison	169
5.4 Conclusions	178

	Page
CHAPTER 6 RELIABILITY OF THE EXTRA STAGE CUBE INTERCONNECTION NETWORK	180
6.1 Introduction	180
6.2 Fault Model	181
6.3 Study of Multiple-Fault Tolerance	181
6.3.1 Terms with Input or Output Stage Box Faults	187
6.3.2 Term without Input or Output Stage Box Faults	192
6.3.3 Solution for Two Faults	200
6.4 Conclusions	200
CHAPTER 7 PROPERTIES OF AN ENHANCED EXTRA STAGE CUBE INTERCONNECTION NETWORK	202
7.1 Introduction	202
7.2 Eliminating Hardcore	202
7.3 Individual Box Enabling/Disabling	203
7.4 Reliability of the Enhanced Extra Stage Cube Network	208
7.5 Conclusions	211
CHAPTER 8 ANALYSIS OF ALTERNATIVE SWITCHING ELEMENTS	214
8.1 Introduction	214
8.2 Previous Work	216
8.3 Switching Element Model	218
8.4 4x4 Crossbar Node Performance Analysis	222
8.5 4x4 Composite Node Performance Analysis	227
8.5.1 Problem Overview and Notation	228
8.5.2 Passing Two Messages	236
8.5.3 Passing Three Messages	239
8.5.4 Passing Four Messages	243
8.5.5 Permutation Delay Probability	251
8.5.6 Summary	251
8.6 Network Implementation and Performance	254
8.7 Conclusions	259

	Page
CHAPTER 9 STUDY OF IMAGE CONTOUR EXTRACTION WITH NETWORK AND SYSTEM IMPLICATIONS	261
9.1 Introduction	261
9.2 Contour Extraction	262
9.3 Overlapped Subimage Method	265
9.4 Non-Overlapped Subimage Method	267
9.4.1 EGT Algorithm	269
9.4.2 Contour Tracing Algorithm	274
9.5 Example of Non-Overlapped Method Performance	292
9.6 Analysis of Non-Overlapped Method	301
9.7 Implications for Network Design	305
9.8 Implications for System Design	308
9.9 Conclusions	311
CHAPTER 10 CONCLUSIONS	313
LIST OF REFERENCES	317
APPENDICES	
Appendix A - Program to Count Lossy Pairs with Stage Bypassing	334
Appendix B - Program to Count Lossy Pairs with Box Bypassing	342
VITA	351

LIST OF TABLES

Table	Page
4.1 Summary of information on the multiple fault tolerance of the ESC network. "FFIC" stands for fault-free interconnection capability.	98
4.2 One-to-one routing tags for the ESC. The symbol "x" represents either 0 or 1.	107
4.3 Broadcast routing tags for the ESC. The symbol "x" represents either 0 or 1.	108
5.1 Number of paths in the Gamma network for possible values of $(D - S)$ modulo N [PaR82].	148
5.2 Summary of fault tolerance information for the networks surveyed. "SE" is an abbreviation for switching element.	170
5.3 Comparison of surveyed networks with the ESC. Entries give the relationship between the network in question and the ESC as regards a particular attribute. "SE" is an abbreviation for switching element	172
5.4 Fault tolerance capabilities of the networks using the ESC network fault model and fault tolerance criterion.	176
7.1 Ratio of lossy pairs involving a stage n or 0 box fault to those not, and the corresponding percentage of lossy pairs involving a stage n or 0 box fault, given stage bypassing.	205

Table	Page
7.2 Ratio of probability of loss of fault-free interconnection capability using stage bypassing versus box bypassing, for BB- and LB-lossy pair types. The probability of loss of fault-free interconnection capability due to LL-lossy pairs does not depend on bypassing method.	210
8.1 A complete listing of the ways conflict can occur in the 4×4 composite node as a function of m , the number of messages initially at the inputs of the node. The associated probability term is given after each table entry corresponding to a case of the performance analysis.	220
8.2 Summary of transit time performance and incremental delay characteristics for the crossbar and composite nodes.	252
8.3 Comparison of permuting capabilities of ESC networks implemented with the maximum number of composite and crossbar nodes (stage n bypassed) for various values of N	258
9.1 Computed parameters for the figure of merit values shown in Figure 9.12.	298

LIST OF FIGURES

Figure	Page
1.1 (a) Generalized wing and engine nacelle showing relevant dimensions to determine engine nacelle installation. (b) Plot of nacelle installations for a number of transport jet aircraft [RuT83].	2
1.2 Position of nacelle using the best (baseline) wind tunnel technology and a computationally-derived installation close-coupled with the wing [RuT83].	4
1.3 Plot of several computationally-derived nacelle installations in comparison to wind tunnel methodology along with a sketch of each [RuT83].	5
1.4 Processing element-to-processing element (PE-to-PE) architecture.	12
1.5 Processor-to-memory (P-to-M) architecture [Sie85].	13
2.1 Block diagram of PASM.	24
2.2 PASM Parallel Computation Unit.	25
2.3 PASM Micro Controllers.	27
2.4 Reconfigurable bus for linking Micro Controller processors and memory modules. Each box can be set to "through" or "short."	29
2.5 PASM Micro Controllers configured with reconfigurable bus.	30
2.6 Organization of the PASM Memory Storage System for $N = 32$ and $Q = 4$, where "MSU" is Memory Storage Unit, "MC" is Micro Controller, and "PCU" is Parallel Computation Unit.	32

Figure	Page
2.7 Block diagram of proposed PASM prototype.	35
2.8 Block diagram of the Numerical Aerodynamic Simulation Facility [Bur79].	37
2.9 General organization of the Flow Model Processor [Bur79].	38
2.10 Block diagram overview of TRAC.	44
2.11 Block diagram of a typical STARAN system [Bat74].	47
2.12 STARAN array module [Bat76].	49
2.13 System architecture of PUMPS [BFH82].	52
3.1 The Generalized Cube network with $N = 8$, and the four states of an interchange box.	57
3.2 Relationship of the Generalized Cube network to an n -dimensional cube for $N = 8$, i.e., $n = 3$	60
3.3 The Extra Stage Cube network with $N = 8$	63
3.4 (a) Detail of interchange box with multiplexer and demultiplexer for enabling and disabling. (b) Interchange box enabled. (c) Interchange box disabled.	65
3.5 The path from input 2 to output 1 in the ESC for $N = 8$, when stage n is disabled and stage 0 is enabled. Stage n and 0 multiplexer and demultiplexer settings are shown explicitly [Sie85].	66
3.6 The path from input 2 to output 1 in the ESC for $N = 8$, when stage n is enabled and stage 0 is disabled. Stage n and 0 multiplexer and demultiplexer settings are shown explicitly [Sie85].	68
4.1 The two paths in the ESC network for $N = 8$ between input 1 and output 4 when both stages n and 0 are enabled. Stage n and 0 multiplexer and demultiplexer settings are shown explicitly.	75

Figure	Page
4.2 ESC network with $N = 8$ and a faulty stage 2 link (indicated by the broken line) showing the two paths from input 1 to output 4, one of which is fault-free.	78
4.3 The two paths from input 0 to outputs 2, 3, 6, and 7 in the ESC network for $N = 8$. The bold lines indicate one complete broadcast path, the dashed lines denote the other [Sie85].	80
4.4 ESC network with $N = 8$ and faults (2,2) and (1,4) (indicated by broken lines), which do not cause a loss of fault-free interconnection capability.	88
4.5 ESC network with $N = 8$ and faults (2,5), (1,4), and (1,6) (indicated by broken lines), which cause a loss of fault-free interconnection capability. The affected network inputs and outputs are circled.	91
4.6 ESC network with $N = 8$ and faults (2,2) and (1,5) showing the primary broadcast path from input 2 to outputs 1, 3, 5, and 7. This path contains fault (2,2).	94
4.7 ESC network with $N = 8$ and faults (2,2) and (1,5) showing the secondary broadcast path from input 2 to outputs 1, 3, 5, and 7. This path contains fault (1,5).	95
4.8 ESC network with $N = 8$ and faults (2,2) and (1,5) showing a fault-free broadcast path from input 2 to outputs 1, 3, 5, and 7 that combines portions of the primary and secondary broadcast paths.	96
4.9 Path used when routing from input 1 to output 4 in a fault-free ESC with $N = 8$	99
4.10 Broadcast path for broadcasting from input 5 to outputs 2, 3, 6, and 7 in a fault-free ESC with $N = 8$	101
4.11 Generalized Cube network with $N = 8$ partitioned into two subnetworks of size $N' = 4$ based on the high-order bit position. The A and B labels denote the two subnetworks.	110
4.12 ESC network with $N = 8$ partitioned into two subnetworks of size $N' = 4$ based on the high-order bit position. The A and B labels denote the two subnetworks.	112

Figure	Page
4.13 Generalized Cube network with $N = 8$ partitioned into two subnetworks of size $N' = 4$ based on the low-order bit position. The A and B labels denote the two subnetworks.	114
4.14 A variation on the ESC network, shown for $N = 8$, that allows partitioning into two subnetworks of size $N' = 4$ based on the low-order bit position. The A and B labels denote the two subnetworks.	115
4.15 ESC network with $N = 8$ and faults (2,2) and (1,4) (indicated by broken lines) showing all fault-free primary paths for the permutation mapping input x to output $(x + 2)$ modulo N , for $0 \leq x < N$	119
4.16 ESC network with $N = 8$ and faults (2,2) and (1,4) (indicated by broken lines) showing the secondary paths corresponding to faulty primary paths for the permutation mapping input x to output $(x + 2)$ modulo N , for $0 \leq x < N$	120
5.1 The Modified Baseline network for $N = 8$	126
5.2 An Augmented Delta network constructed from a $b^n \times b^n$ Delta network [Dia81].	128
5.3 An Omega network for $B = 2$, $N = 8$, and $m = 3$	132
5.4 (a) A Multipath Omega network for $N = 16$ corresponding to the pseudofactorization $\langle 4,2,2,4 \rangle$. (b) Its associated redundancy graph [PaL83a].	133
5.5 Several redundancy graphs [PaL83a].	135
5.6 The F-network for $N = 8$	138
5.7 Routing algorithm for the F-network.	140
5.8 The Inverse Augmented Data Manipulator network for $N = 8$	141
5.9 The Enhanced Inverse Augmented Data Manipulator network with half links for $N = 8$	143

Figure	Page
5.10 The Gamma network for $N = 8$ [PaR82].	146
5.11 Beneš network for $N = 8$	150
5.12 Fault-tolerant Beneš network for $N = 8$	153
5.13 Switching element of an ACN network.	156
5.14 (a) Connections to stage 0 switches in an ACN. (b) Connections from stage $n - 1$ in an ACN. (c) Connections from a conjugate switch pair in an ACN.	157
5.15 Flow chart for one ACN routing strategy.	159
5.16 Flow chart for another ACN routing strategy.	160
5.17 A simple single-stage β -network.	162
5.18 The β -graph of the example single-stage β -network [ShH80].	164
5.19 Two circuit partitions of the example β -network [ShH80].	165
5.20 The two circuit adjacency graphs for the two circuit partitions, respectively [ShH80].	167
5.21 Two Eulerian circuits of the β -graph of the example β -network [ShH80].	168
6.1 Event space for classes of physical fault groupings in the ESC. The region characterized by loss of fault-free interconnection capability is enclosed by the bold line (drawing not to scale).	183
6.2 ESC network with $N = 8$, a given faulty box (indicated by the bold line), and all network components in stages 2 and 1 with which it can form a lossy pair (indicated by the broken lines).	193

Figure	Page
7.1 ESC network configured with two input and two output ports per device. (a) and (b) Detail of bypass circuitry. (c) and (d) Enabled and disabled input box, respectively. (e) and (f) Enabled and disabled output box, respectively.	204
7.2 $P(\text{loss} N, 2)$ as a function of network size for various values of $P(\text{BF} F)$. Solid lines are for stage bypassing; dashed lines are for box bypassing.	212
8.1 A 4×4 composite switching element, or node.	219
8.2 (a) A 4×4 crossbar switching element, or node. (b) Crosspoint.	221
8.3 (a) Example of a subsequent delay. (b) Example of a subsequent conflict. (c) Example of an arbitration delay.	231
8.4 Message movement in a composite node for Table 8.1 entry IV.B.2.b. Messages are indicated by the letters A, B, C, and D. (a) Initial message position. A and B conflict in level 1. (b) Position after one time step. A and C conflict in level 2. (c) Position after two time steps. B experiences arbitration delay. (d) Position after three time steps.	233
8.5 Message movement in a composite node for Table 8.1 entry IV.B.2.c. Messages are indicated by the letters A, B, C, and D. (a) Initial message position. A and B conflict in level 1. (b) Position after one time step. A and C conflict in level 2. (c) Position after two time steps. B and C subsequently conflict in level 2. (d) Position after three time steps.	235
8.6 Message movement in a composite node corresponding to one element in the set of message flow patterns covered by $P(t=3, \text{case } 3 m=4)$. (a) Initial message position. A and B conflict in level 1. (b) Position after one time step. A and C conflict in level 2. (c) Position after two time steps. No remaining conflicts.	245

Figure	Page
8.7 Message movement in a composite node corresponding to one element in the set of message flow patterns covered by $P(t=4, \text{ case } 4 \mid m=4)$. (a) Initial message position. (b) Position after one time step. (c) Position after two time steps. (d) Position after three time steps.	249
8.8 A composite node labeled for use in construction of an ESC network.	255
8.9 Construction of an ESC with $N = 8$ and 4×4 switching elements.	257
9.1 Two 8-adjacent 1s and two non-4-adjacent 0s.	264
9.2 (a) Data allocation for a $R \times R$ image using N PEs. (b) Data transfers needed to apply Sobel edge operator.	268
9.3 Sobel operator algorithm defined for a subimage.	270
9.4 Parallel algorithm for EGT at PE_i	273
9.5 (a) Naming convention for the neighbors of the center pixel in a 3×3 window. (b) Example showing start of tracing.	278
9.6 Example of Phase I contour tracing for a 10×20 image. The triple (i,x,y) represents the i - x - y coordinates of the pixel.	282
9.7 Contours found by Phase I row scans for a 30×20 sample image.	283
9.8 Contours found by the Phase I column scan for the image of Figure 9.7.	284
9.9 Example to illustrate Phase II activity, protocol, and mechanisms. End point coordinates are given where (i,x,y) represents the i - x - y coordinates of the pixel.	289
9.10 Section of a poorly illuminated circuit board overlaid with a 16×16 pixel grid.	293

Figure	Page
9.11 A binary image obtained by thresholding Figure 9.10 with a single threshold of 153.	294
9.12 The EGT merit value graphs for the 64 subimages in the upper left corner of Figure 9.10. The horizontal axis for each graph is gray value (0 to 255) and the vertical axis is the threshold merit value (0 to 32).	296
9.13 Binary image resulting from EGT-based segmentation of Figure 9.10.	299
9.14 Contours extracted from Figure 9.10.	300

ABSTRACT

Adams, George Bunch, III. Ph.D., Purdue University, December 1984. A FAULT-TOLERANT INTERCONNECTION NETWORK AND IMAGE PROCESSING APPLICATIONS FOR THE PASM PARALLEL PROCESSING SYSTEM. Major Professor: Howard Jay Siegel.

The demand for very high speed data processing coupled with falling hardware costs has made large-scale parallel and distributed computer systems both desirable and feasible. Two modes of parallel processing are single instruction stream - multiple data stream (SIMD) and multiple instruction stream - multiple data stream (MIMD). PASM, a partitionable SIMD/MIMD system, is a reconfigurable multimicroprocessor system being designed for image processing and pattern recognition. An important component of these systems is the interconnection network, the mechanism for communication among the computation nodes and memories. Assuring high reliability for such complex systems is a significant task. Thus, a crucial practical aspect of an interconnection network is fault tolerance.

In answer to this need, the Extra Stage Cube (ESC), a fault-tolerant, multistage cube-type interconnection network, is defined. The fault tolerance of the ESC is explored for both single and multiple faults, routing tags are defined, and consideration is given to permuting data and partitioning the ESC in the presence of faults. The ESC is compared with other fault-tolerant multistage networks. Finally, reliability of the ESC and an enhanced version of it are investigated. ()

A knowledge of the performance of various switching element designs is important to the engineering of interconnection networks. Typically, networks proposed for parallel systems have been designed with two-input/two-output switches. VLSI technology allows implementation of complex circuits as a single device. The performance of four-input/four-output switches under various message loading conditions is analyzed and their use in the ESC considered.

Finally, a parallel digital image processing scenario for implementation on a computer system such as PASM is analyzed. Contour extraction is chosen as the focus because it is a key step in many applications and presents a multifaceted challenge to a parallel computer. Issues studied include parallel formulation of the constituent algorithms, mapping the algorithms and sizing the machine, quality of results, and implications for network design and system architecture.

CHAPTER 1

INTRODUCTION

1.1 Motivation

Consider the experience of the Boeing Commercial Airplane Company in a project to improve the propulsive efficiency of the Boeing 737 aircraft [RuT83, TiC84]. The 737 was originally designed with low-bypass ratio turbojet engines mounted below the wings. Advances in technology have led to high-bypass ratio turbofan engines, which are more fuel efficient than the turbojet engines they replace, yet, are inherently larger in diameter.

Conventional aerospace engineering practice prescribes a certain separation between an engine nacelle, or housing, and wing so as to avoid excessive drag due to airflow interference between the two structures. Figure 1.1(a) shows a generalized wing and nacelle with the relevant dimensions indicated. Figure 1.1(b) shows the state of the art concerning nacelle installation, achieved through years of wind tunnel testing. Aircraft designers found that a nacelle positioned so as to be above the dotted line in Figure 1.1(b) gave rise to excessive drag (three to five percent of the total aircraft drag). The precise nature of the drag was unknown. Thus, the dotted line represents both a family of closest installations for nacelle and wing, based upon conventional design as supported by wind tunnels, and a baseline from which to measure new design techniques.

Baseline Via Wind Tunnel Test Methodology

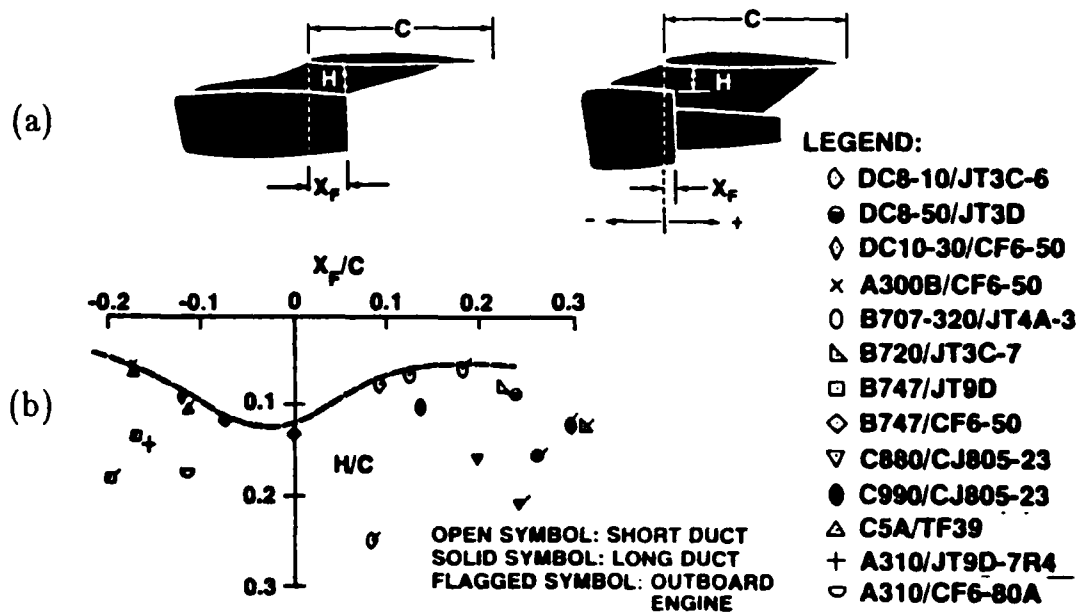


Figure 1.1 (a) Generalized wing and engine nacelle showing relevant dimensions to determine engine nacelle installation. (b) Plot of nacelle installations for a number of transport jet aircraft [RuT83].

Using conventional design for the 737 engine refit project results in the high-bypass engines contacting the runway (see Figure 1.2) due to their large diameter and the prescribed separation between wing and nacelle. To eliminate this problem the landing gear would have to be lengthened, requiring redesign and sharply reducing the possible attractiveness of the refit to owners of existing 737 aircraft due to the additional expense to install new landing gear. Further, the longer landing gear would have a greater weight, offsetting some of the fuel efficiency gain of the new engines.

The relatively recent availability of sufficiently powerful computers made study of engine/wing interference drag, via aerodynamic simulation, feasible. This research revealed the source of the drag [daC78] and the design solution: proper choice of the shape of the nacelle and nacelle support strut. Computer simulation guided the designers in the 737 project to nacelle and strut shapes that afforded acceptably low drag and achieved adequate ground clearance with the existing landing gear while being compatible with housing the engine and its accessories (see Figure 1.2). An additional benefit of the close-coupled installation is the reduced size, and hence, weight, of the nacelle support strut.

Flight testing this year of the refitted 737 (known as the 737-300) has shown the interference drag due to the nacelle installation to be much less than one percent of total aircraft drag [Tin84]. Figure 1.3 shows the relationship of the computationally-derived nacelle installation for the 737-300 (as well as for similar projects for the 707, 757, and 767 aircraft) to that possible with conventional wind tunnel methodology. It also shows the range of nacelle and strut shapes used. These computationally derived nacelle shapes and installations lie above the dotted line shown in the plot of Figure 1.3 indicating a close installation, yet they do not incur high drag.

Impact of Close Nacelle Coupling on the 737-300 Design

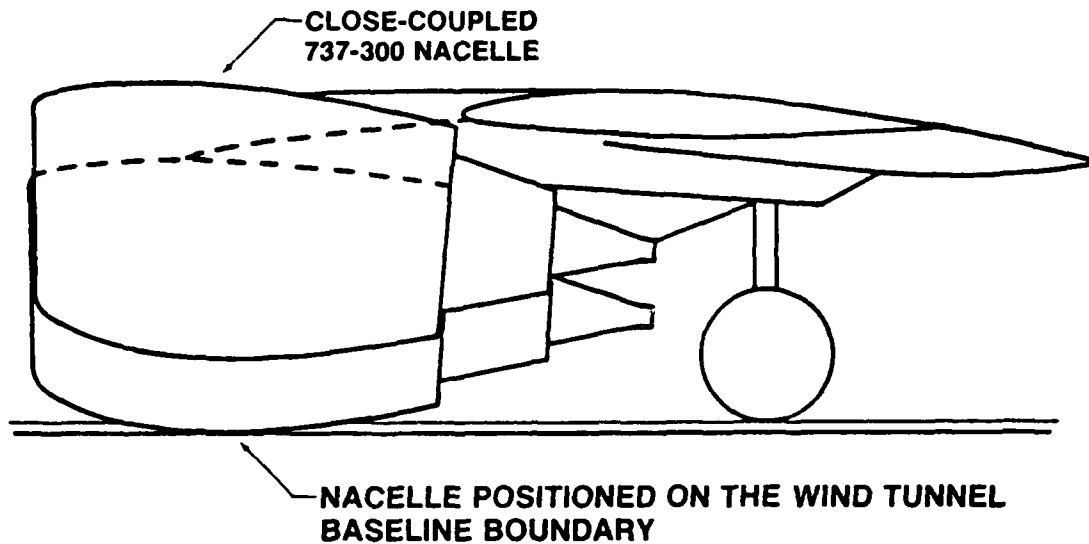


Figure 1.2 Position of nacelle using the best (baseline) wind tunnel technology and a computationally-derived installation close-coupled with the wing [RuT83].

Computationally Derived Close-Coupled Nacelle Positions

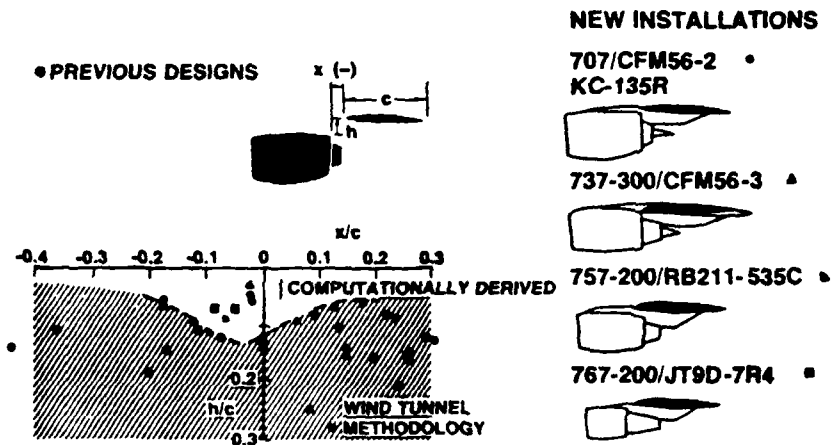


Figure 1.3 Plot of several computationally-derived nacelle installations in comparison to wind tunnel methodology along with a sketch of each [RuT83].

The alternative to computer simulation, empirical efforts to develop low drag nacelle installations via wind tunnel experimentation, has not been too successful [RuT83] for several reasons. One reason is that wind tunnels readily provide information about total drag, but not the drag due to individual airframe components. Another is that it is not always possible to measure wind tunnel air flow phenomena in fine detail. The development of the 737-300 would not have been accomplished without high-speed computers.

High-speed computation is vital to the future of many human endeavors in addition to aircraft design. Continued progress in such disciplines and activities as theoretical physics and chemistry, flight simulation, fusion energy research, image generation and processing, integrated circuit design and simulation, hydrocarbon exploration and reservoir modeling, continuous speech recognition, structural analysis, and weather forecasting depends upon the continuing availability of yet faster computers. There are important problems in each of these areas that can neither be feasibly solved using available computers nor be solved without computers.

Modern high-performance computers deliver up to roughly several hundred million floating point operations per second (MFLOPS) when executing actual applications programs; speeds typically average more nearly 10 to 50 MFLOPS. Careful programming can increase these achieved rates, but only by a factor of two to four. Over the next ten years, a computing performance increase by a factor near one thousand will be needed to sustain normal progress in many disciplines [AdD84]. This computational need cannot be dismissed as the result of inadequate algorithm design. Continued progress with algorithms will reduce the execution time of many problems, but faster hardware will still be essential. The demand for increased computational power is growing rapidly

and will continue to grow for the long term future.

The uniprocessor design of computers is today approaching fundamental physical limits to ultimate performance. While circuit switching speeds and circuit packing density will continue to improve, the impressive historic rate of increase is likely to slow. It is unlikely that uniprocessor technology will provide sustained speeds beyond 1000 MFLOPS in this century. Yet, this is the realm of computational power that is needed. An opportunity to achieve greater processing speeds lies with computer systems employing multiple processing elements acting together. The truly large gains will be made if hundreds and even thousands of processors can be made to work effectively in concert.

The demand for very high speed computing coupled with falling hardware costs has made large-scale parallel and distributed computer systems both desirable and feasible. An important component of these systems is the interconnection network, the mechanism for information transfer among the computation nodes and memories. Assuring high reliability for such complex systems is a significant task. Thus, a crucial practical aspect of an interconnection network is fault tolerance. Study of a fault-tolerant multistage interconnection network is one of three topics in this work

A knowledge of the performance of various network switching element designs is important to the engineering of interconnection networks. VLSI technology opens up wider possibilities for network implementation by allowing more complex, sophisticated switching elements. An analysis of alternative switching element designs suited for VLSI implementation is a second area addressed in this work. The switching elements investigated could replace the simpler ones used in numerous interconnection networks, including

the one studied here, lowering cost and/or improving network performance.

Finally, the interaction of application programs and parallel/distributed systems is a fundamental question. Detailed investigation of a particular application from a problem domain can guide many aspects of system design as well as illuminate the issue of mapping the application to the machine. The third aspect of this work is an investigation of digital image contour extraction using a parallel computer.

1.2 Overview

The approach taken in this work attempts to be cognizant of engineering considerations for building a parallel computer system. For example, not only must an interconnection network have desirable theoretical properties, these properties should be feasible to attain and/or use. The intent is to attain useful knowledge for design of parallel and distributed computer systems.

Chapter 2 surveys five existing or proposed parallel processing systems that use multistage interconnection networks in their designs. These systems could potentially incorporate the fault-tolerant interconnection network developed and studied in later chapters. The information presented in this chapter is intended to provide a context in which such networks can be viewed.

The fault-tolerant interconnection network that is the focus of much of this work is the Extra Stage Cube (ESC), which derives from the Generalized Cube network. Chapter 3 is the first of four chapters to deal with the ESC network. In it the Generalized Cube network is defined and its basic properties stated, then the ESC is defined.

Properties of the ESC are set forth in Chapter 4. Its single fault tolerance is established and capacity for coping with multiple faults studied. Routing tags for network control under both fault-free and faulted conditions are defined. Partitioning the network and permuting data using it are also discussed.

Chapter 5 reviews the state of the art in fault-tolerant multistage interconnection networks. The characteristics of these networks are described and the nature of their fault tolerance discussed. Each network is compared with the ESC.

With the fault-tolerant capabilities of the ESC determined, Chapter 6 presents an analysis of ESC reliability. Reliability is measured as the probability that there exists at least one path between any network input and output. A exact solution for the case of two faults is developed based on the ESC fault model stated in Chapter 4.

Consideration of the results in Chapter 6 shows possible areas for improvement in the ESC design and operational protocol. In Chapter 7, an enhancement of the basic ESC topology is described. This provides increased fault tolerance with only a small increase in system hardware complexity; no logic need be added to the ESC. The effect of the enhancement on ESC properties and reliability is investigated. Large networks are shown to benefit more than small ones from this modification.

Chapter 8 presents a performance analysis and comparison of two switching elements suitable for use in the ESC and many other interconnection networks. These switching elements are alternatives to the traditional interchange box and are suited to very large-scale integration (VLSI) manufacture.

PASM is a parallel computer system intended to be suited for image processing and pattern recognition tasks. To insure that the design architecture can meet this goal, it is useful, probably even necessary, to consider selected image processing tasks to learn what is required of system hardware for their execution in a manner satisfactory to users. In Chapter 9 a scenario for image contour extraction based on parallel computation is developed and used to explore the advantages of parallel computation and issues in parallel computer design. Parallel forms of edge-guided thresholding and contour tracing algorithms are constructed and analyzed to highlight important aspects of the scenario. The implications that the scenario has for parallel computer architecture are considered, including interconnection network design. Various important system attributes are identified and described.

The remaining sections of this chapter present the basic vocabulary and definitions needed in subsequent chapters. Descriptions of the major parallel computer architecture classes and some of their important characteristics are included. Fundamental interconnection network terminology is defined.

1.3 Parallel Computer System Terminology

An *SIMD* (Single Instruction Stream - Multiple Data Stream) [Fly66] machine typically consists of a control unit, N processors, N memory modules, and an interconnection network. (e.g., Illiac IV [BDM72]). The control unit broadcasts instructions to all of the processors, and all active processors execute the same instruction at the same time. Thus, there is a single instruction stream. Each active processor executes the instruction on data in its own associated memory module. Hence, there are multiple data streams.

The interconnection network, sometimes referred to as an alignment or permutation network, provides a communications facility for the processors and memory modules [Sie79b, Sie85]. Illiac IV [BDM72] and the Massively Parallel Processor (MPP) [Bat80] are examples of SIMD systems.

Many architectural variations are possible within the class of SIMD machines. One subclass of SIMD machines are those with the *processing element-to-processing element (PE-to-PE)* organization. Figure 1.4 shows this type of architecture. In this scheme, each processor is paired with a memory module to form a *processing element (PE)*. The interconnection network need only support unidirectional information transfer, as each PE has access to both a network input and output to transmit and receive information, respectively. The network may be able to connect a given PE to all or a subset of the other PEs. If two PEs cannot be directly connected through the network, then indirect transfer of information through one or more PEs is necessary. Each successive transfer is accomplished by an additional pass through the network. The Illiac IV computer used the PE-to-PE organization and had an interconnection network that allowed direct connection between a given PE and only its four nearest neighbors, where PEs were arranged in a square mesh fashion [BBK68, BDM72].

Another SIMD machine subclass is the *processor-to-memory (P-to-M)* structure, which utilizes an interconnection network placed between the processors and memories. Figure 1.5 shows this architecture. Note that the number of processors and memories need not be the same. The interconnection network must support bidirectional information transfer in these machines. A memory module is said to be common to two processors if both can access it directly through the network. Two processors can thus communicate directly

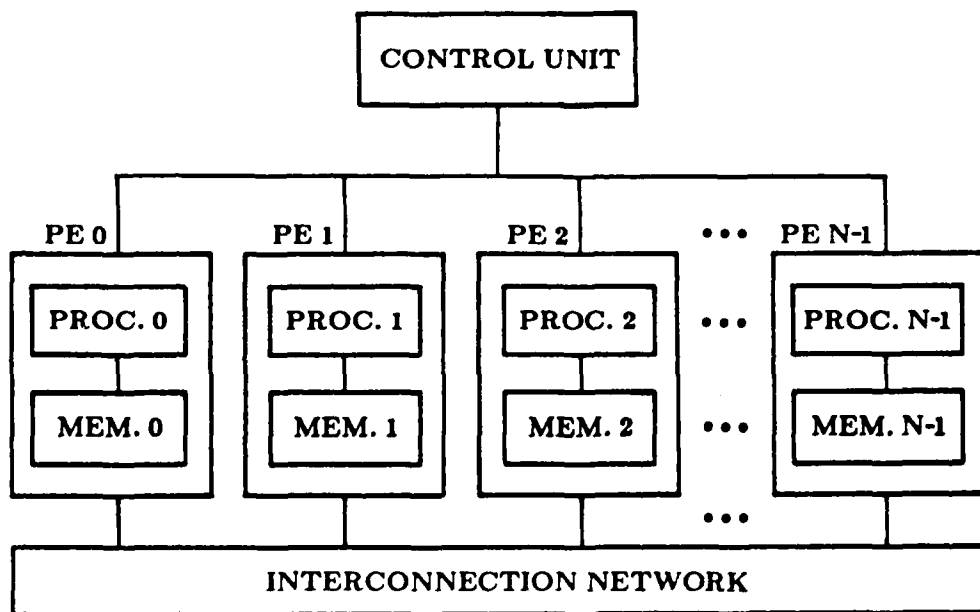


Figure 1.4 Processing element-to-processing element (PE-to-PE) architecture.

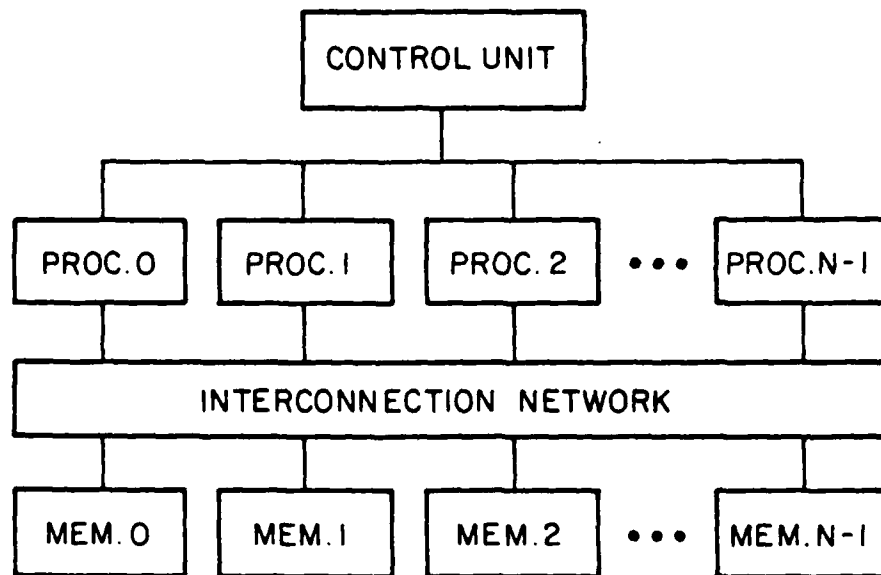


Figure 1.5 Processor-to-memory (P-to-M) architecture [Sie85].

through a memory module they have in common. If no such memory module exists, information can be passed by intermediate processors through intermediary memory modules. The P-to-M structure is used in the Texas Reconfigurable Array Computer (TRAC) [KPL80, PKM80, SUK80]; the number of memories is larger than the number of processors for this machine.

An *MSIMD* (multiple-*SIMD*) machine is a parallel processing system which can be dynamically reconfigured to operate as one or more independent *SIMD* machines of potentially various numbers of processors and memory modules. Each independent *SIMD* machine is a *partition* of the *MSIMD* system and is referred to as a *virtual SIMD* machine. An *MSIMD* system typically consists of N processors, N memory modules, an interconnection network, and Q control units, where $1 < Q < N$. Proposed systems capable of *MSIMD* mode operation include MAP [Nut77] and PASM [SSK81]. Illiac IV was originally intended to be an *MSIMD* system [BBK68]. *MSIMD* systems can be of either the PE-to-PE or P-to-M type.

Possible advantages of a *MSIMD* system relative to an *SIMD* system with a similar number of PEs include the following [SSK81].

1. **Fault detection** - For situations requiring high reliability, three or more partitions can process the same data identically and compare results.
2. **Fault tolerance** - If a single PE fails, only those virtual *SIMD* machines (partitions) that include the failed PE are affected. The rest of the system can continue to function as before.

3. Multiple simultaneous users - Since there can be multiple independent virtual SIMD machines, there can be multiple simultaneous users without having to support multitasking.
4. Program development - Rather than debugging an SIMD program on the entire system, a user can select a small partition for program testing.
5. Efficiency - If a task requires only a subset of the available PEs, the other PEs can be used for a different task.
6. Subtask parallelism - Two or more independent SIMD subtasks that are part of the same job can be executed in parallel, sharing results if necessary.

An *MIMD* (*Multiple Instruction Stream - Multiple Data Stream*) machine [Fly66] typically consists of N processors and M memories, where each processor can execute an independent instruction stream. As with SIMD architectures, there are multiple data streams and an interconnection network. Thus, there are N independent processors which can communicate among themselves. These systems can be organized as either PE-to-PE or P-to-M. There may be a coordinator unit to help orchestrate processor activity. Cm* [SFS77] and C.mmp [WuB72] are two MIMD systems that have been constructed.

A *partitionable SIMD/MIMD* machine is a parallel computer that can be reconfigured as one or more independent virtual SIMD and/or MIMD machines of various sizes [SMM79]. Such a machine has the same structure as an MSIMD system, but the processors are capable of fetching, decoding, and

executing their own instruction streams in addition to acting on an instruction stream from a control unit. Thus, each partition that can be formed by the system can function as either a virtual SIMD or MIMD machine, and the mode of operation within a partition can change over time. PASM [SSK81] and TRAC [KPL80, PKM80, SUK80] are examples of partitionable SIMD/MIMD systems.

The processors, memories, and interconnection network of a partitionable SIMD/MIMD computer can be organized in various ways, as discussed for SIMD machines. PASM uses the PE-to-PE organization, while TRAC uses P-to-M. The advantages of an MSIMD system relative to SIMD machines are available with partitionable SIMD/MIMD systems. It is possible to switch between SIMD and MIMD modes to best accommodate successive algorithms or successive phases of one algorithm.

1.4 Interconnection Network Terminology

An *interconnection network* is a device designed to provide high-speed communication among a set of processors that typically are physically close and more or less closely coupled in operation. It is a key element in computers of each of the parallel computer classes discussed in Section 1.2. For convenience, the term "network" will often be substituted for the more cumbersome "interconnection network" in the following. Networks are comprised of a collection of switches, or *switching elements*, and *links*, or wires. Switching elements are also referred to as *nodes*. A network may consist of a single *stage*, or bank, of switches, or multiple stages connected by links. A *single stage network* utilizes only one stage of switches, and data sent by a device using the network may have to pass through the network repeatedly to

reach its intended destination. A *multistage network* is constructed from two or more stages of switches, and, typically, data can be sent to the desired destination via one pass through the network.

The *hardware complexity* of an interconnection network is a measure of the number of components required for its construction. This parameter is frequently used to make general comparisons between networks because it gives an approximate indication of relative implementation cost. An asymptotic complexity measure is typically used for this purpose since it can clearly express basic trends. Let f be a function representing the number of network components as a function of network size. Then, $f(x)$ is *of order* $g(x)$ (written $O(g(x))$) if there exist constants c and x_0 such that

$$f(x) \leq c g(x)$$

for $x \geq x_0$ [Knu76, AHU74].

A network that supports information flow in only one direction through the stage(s) is *unidirectional*. Each device using such a network must have access to both a network input and output in order for communication between two devices to be possible. Network inputs and outputs are generically referred to as *ports*. For a unidirectional multistage network, the stage to which input ports are connected is the *input stage*, and the stage connected to the output ports is the *output stage*. A *bidirectional* network allows data to pass through the stage(s) in either direction. There is no distinction of network ports vis-à-vis the classes input and output for bidirectional networks; a device need have access to only one port to communicate in this case.

The *topology* of a network is the pattern of connections in the structure of the network. It is determined by the nature of the switching elements, the

connections between network ports and switching elements, and the connections between stages of switching elements (for multistage networks). Network topology is often used to compare different networks [McS82b, Thu74, WuF80]. This is because such comparisons are independent of the particular implementation of a network. The network analyses performed in the following chapters are based on network topology. Thus, the results obtained are not specific to any particular hardware technology.

Patterns of data flow through a network can be classified into three basic categories. With *one-to-one connections* information is passed from one network port, the *source*, to another network port, the *destination*. The exact route taken by the information is its *path*. Some networks provide multiple paths between a source and destination. Often, information flow from one source to two or more destinations is supported by a network. Such a transfer is termed a *broadcast connection*, and the route taken by the information is a *broadcast path*. Finally, consider a set of non-intersecting one-to-one connections, that is, a set such that no two one-to-one connections have the same source or destination. If these connections can be created simultaneously within a network, a *permutation connection* results.

Many networks will not support all permutation connections. A permutation is not supported if there is a *conflict* between two of the one-to-one connections, i.e., both connections require a single output of some switching element. Two such paths are said to *contend* for the switching element output, and if one path is given control of the switch, the other experiences *blocking*. A network that does not pass some permutations due to conflict between paths is a *blocking network*. Switching elements can be viewed as small interconnection networks and, hence, may also be characterized as

blocking if they do not support all permutation connections from their inputs to outputs.

In addition to various types of data flow patterns that may be allowed by a network, there are two basic variations of information transfer protocol. One is *circuit switching*. In this protocol a complete path from the source to the destination is established before information transmission begins. If the network provides multiple paths they may be searched to find one that is not blocked. The complete path is maintained until the data transfer ends, then it is relinquished. Complete broadcast paths are used to support circuit-switched broadcasting.

The other protocol is *packet switching*. In this case, a complete message is subdivided into a sequence of message fragments called *packets*. Each packet carries information that directs it through the network. The packets are presented one at a time to the network by the source. In a multistage network, packets move stage by stage toward their destinations. At each stage the appropriate link to the next stage is determined and its use requested. If the switching element to receive the data in the next stage can accommodate the packet, permission to send is granted to the switching element with the packet. If not, the packet either waits for passage to be granted or is sent on an alternate link (if one exists and is available). Both unidirectional and bidirectional networks can be designed to operate under either circuit or packet switching protocols, or both.

There are three basic types of network control for multistage networks [SiS78]. One is *individual stage control*. With this method a single command signal (which may be several bits) sets the state of all switching elements in a stage. *Partial stage control* is characterized by two or more groupings of

switching elements per stage, each group having separate control. With *individual box* (or switching element) control the state of each network switch can be set independently.

Network control can be implemented in either a centralized or distributed fashion. More typically considered for centralized control are networks with a small number of input/output ports and/or those using individual stage or partial stage control. The advantage of centralized control is that switching element hardware need only include circuitry to handle information flow, not establish or maintain the avenues of flow. The disadvantage is that network performance is limited by the central controller. Central controller speed requirements become more extreme as network size increases (larger N) and for networks in which selection of a route through the network is computationally involved.

Distributed control is particularly well suited to larger networks and those with individual box control. *Routing tags* are a way of encoding information describing a path through a network. They can be generated by the devices using the network, thus distributing control of the network. There are a number of advantages to distributed control. One is that the speed of network path creation is not limited by any potential central controller processing bottleneck, as it might be with centralized control. Another is the ability (for certain networks, e.g., Augmented Data Manipulator network [McS82b]) to reroute information while it is in transit. Switches must be able to change state based on the contents of the routing tags, and there must be provision in the switch for arbitration if there are multiple needs for a single switching element output. These greater hardware requirements may result in high logic-to-pin ratios, allowing switch implementation to take advantage of VLSI.

A disadvantage of distributed network control is the potential higher cost of switching elements.

Dividing a network into independent *subnetworks*, each of which has all the properties of the original network only for a smaller number of ports, is called *partitioning* [Sie80]. Partitioning is useful for networks serving in MSIMD or partitionable SIMD/MIMD systems. A subnetwork can be assigned to each virtual SIMD or MIMD machine. Since subnetworks are disjoint, machine independence is guaranteed.

1.5 Summary

In this chapter the aircraft design example illustrated one specific instance of the need for very high-performance computing and gave motivation to the study of parallel processing computers. The nature and organization of the work presented in this document was outlined. Finally, basic terminology and definitions to be used throughout this work were given.

CHAPTER 2

SURVEY OF SELECTED PARALLEL COMPUTER SYSTEMS

2.1 Introduction

The field of parallel computation research is an active one. Much of the activity revolves around new machine designs, and well over one hundred have been described in the recent literature. It is feasible, therefore, to present only a sample of the work that has been done.

Although the computers described in the following range from systems for which there is only a design, through those that have been prototyped, to a machine that has been sold commercially, all have a common trait. Each utilizes, or is designed to utilize, a multistage interconnection network for communication among various system components. Thus, study of these systems provides information on the environment in which a multistage interconnection network will operate.

These machines also represent a range of architectural ideas for parallel computers intended for signal or image processing, pattern recognition, and related tasks. They thereby illustrate the influence of the application on the machine design for parallel computers. Because the motivation for parallel computation is speed, most parallel computers are optimized for a particular application domain. Consequently, it is important that parallel computer architects study the intended applications of their machines. In this way they can build confidence that the finished machine will perform as desired

(typically, meet execution speed requirements).

2.2 PASM

PASM (*partitionable SIMD/MIMD*) is a special purpose, dynamically reconfigurable, large-scale multimicroprocessor system being built at Purdue University [SMS78, SSK81, SSD84]. Due to the low cost of microprocessors, computer system designers have been considering various multimicrocomputer architectures as a way of achieving high performance. PASM combines the following features: (1) partitionability, allowing operation with many independent SIMD and/or MIMD machines of various sizes; and (2) a design guided by a variety of problems in image processing and pattern recognition.

Figure 2.1 is a block diagram of the basic components of PASM. The heart of PASM is the *Parallel Computation Unit (PCU)* which contains $N = 2^n$ processors, N memory modules, and an interconnection network. The *PCU processors* are microprocessors that perform the actual SIMD and MIMD computations. The *PCU memory modules* are used by the PCU processors for data storage in SIMD mode and both data and instruction storage in MIMD mode. Figure 2.2 shows that the processors and memory modules of the PCU are organized as processing elements. Each memory module consists of a pair of memory units. This double-buffering scheme allows data to be moved between one memory unit and secondary storage (the Memory Storage System) while the processor operates on data in the other memory unit.

The *interconnection network* provides a means of communication among the PCU PEs, which are *physically numbered (addressed)* from 0 to $N-1$. PASM will use either an Extra Stage Cube type network [AdS82d] or an Augmented Data Manipulator type network [AdS80, McS82c, SiM81a]. Both

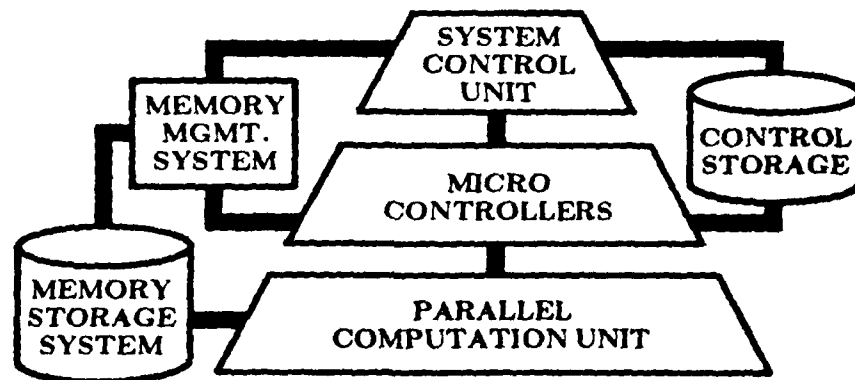


Figure 2.1 Block diagram of PASM.

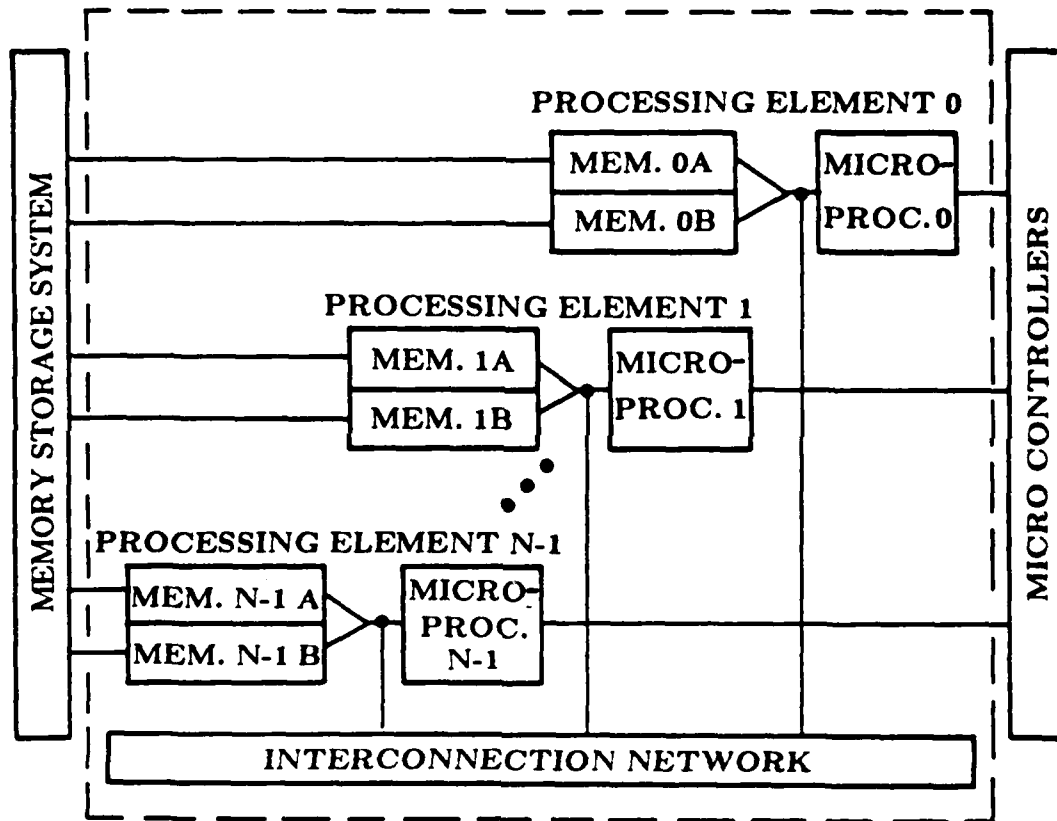


Figure 2.2 PASM Parallel Computation Unit.

consist of $n = \log_2 N$ stages of switches and are controlled by routing tags [Law75, SiM81a]. Both can be partitioned into independent subnetworks of varying sizes, which are powers of two, if all of the PEs in a partition of size $P = 2^p$ have the same value in the low order $n-p$ bit positions of their addresses [Sie80]. Studies are currently being conducted to choose one of these networks to implement in the PASM prototype (e.g., [McS82b]). This work is relevant to that effort.

The *Micro Controllers (MCs)* are a set of $Q = 2^q$ microprocessors, physically numbered (addressed) from 0 to $Q-1$, that are the control units for PCU processors operating in SIMD mode and orchestrate the activities of PCU processors in MIMD mode. Each MC is attached to a memory module which, like the PCU memory modules, consists of a pair of memory units so that memory loading and computations can be overlapped. *Control Storage* contains the programs for the MCs.

Each MC controls N/Q PCU processors. The physical addresses of the N/Q PEs connected to an MC, shown in Figure 2.3, have as their low-order q bits the physical address of the MC, so that the MCs can support system partitioning with either an Extra Stage Cube type or Augmented Data Manipulator type network [SSK81]. Possible values for N and Q are 1024 and 32, respectively. Loading R MC memory modules with the same instructions simultaneously yields a virtual SIMD machine (partition) of size RN/Q , $R = 2^r$ and $0 \leq r \leq q$ [SMS78]. In SIMD mode, the R MCs are synchronized and each MC fetches instructions from its memory module, executing the control flow instructions (e.g., branches) and broadcasting the data processing instructions to its PCU PEs. Similarly, a virtual MIMD machine of size RN/Q results from combining the independent efforts of the PCU processors of R MCs. In both

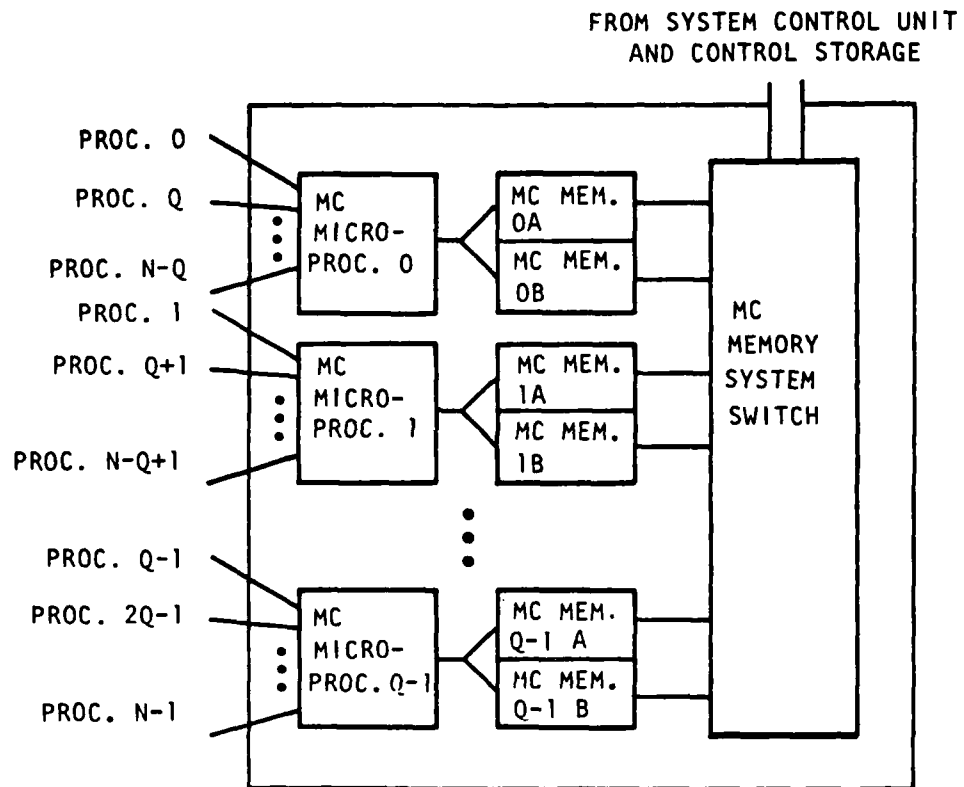


Figure 2.3 PASM Micro Controllers.

cases, the physical addresses of these MCs must have the same low-order $q-r$ bits so that all of the PCU PEs in the partition have the same low-order $q-r$ physical address bits.

The basic MC organization can be enhanced to allow the sharing of memory modules by the MCs in a partition. The MCs can be connected by a shared reconfigurable ("shortable") bus such as described in [ArP76]. This is shown in Figure 2.4 for $Q=8$. The MCs must be ordered on the bus by increasing order of the bit reverse of their addresses so MCs that agree in their low-order address bits can share memory modules. Figure 2.5 depicts the MC processors and memory modules with the reconfigurable bus for $Q=16$. This enhanced MC connection scheme provides more program space for jobs using multiple MCs and a degree of fault tolerance, since known-faulty MC memory modules can be avoided in multiple-MC partitions. These advantages come at the expense of additional system complexity. The use of such a reconfigurable bus to share memories is also discussed in [KaK79].

Within each partition, the PCU PEs have *logical addresses*. Given a virtual machine of size RN/Q , the PEs have logical addresses 0 to $(RN/Q)-1$ (the high-order $r+n-q$ bits of the physical addresses). Similarly, the MCs are assigned logical addresses from 0 to $R-1$ (for $R > 1$, the high-order r bits of the physical address). The PASM language compilers and operating system will translate between logical and physical addresses, so a system user need deal only with logical addresses.

The *Memory Storage System* provides secondary storage for data (SIMD mode) or programs and data (MIMD mode). Multiple devices are used to allow parallel data transfers. The Memory Storage System consists of N/Q independent *Memory Storage Units*, numbered from 0 to $(N/Q)-1$. Each

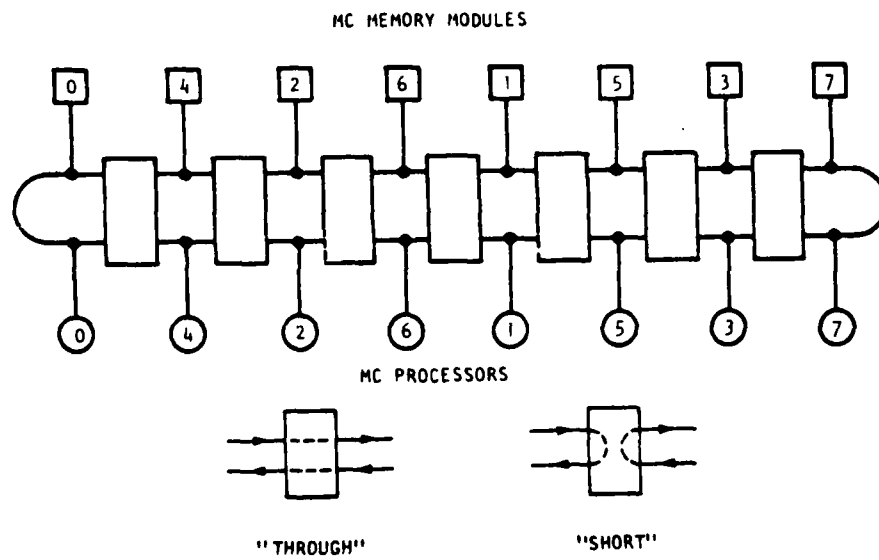


Figure 2.4 Reconfigurable bus for linking Micro Controller processors and memory modules. Each box can be set to "through" or "short."

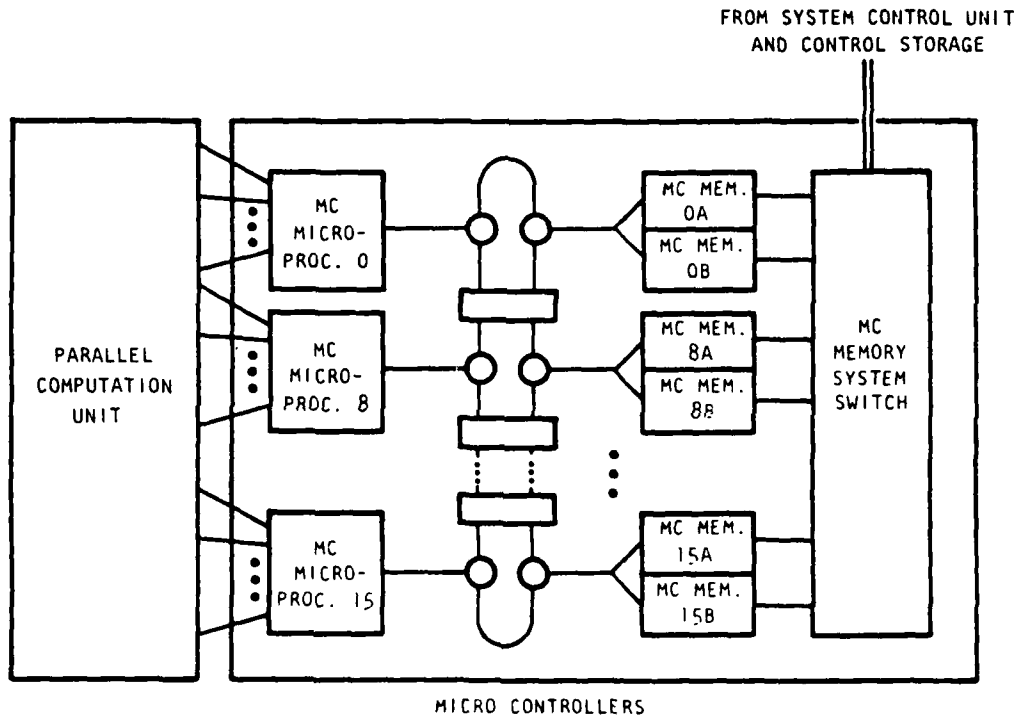


Figure 2.5 PASM Micro Controllers configured with reconfigurable bus.

Memory Storage Unit is connected to Q PCU memory modules. For $0 \leq i < N/Q$, Memory Storage Unit i is connected to those memory modules whose physical addresses are of the form $(Q * i) + k$, $0 \leq k < Q$. Thus, Memory Storage Unit i is connected to the i^{th} PE of each MC as shown in Figure 2.6 for $N = 32$ and $Q = 4$.

The advantages of this Memory Storage Unit connection scheme are that for a partition of size N/Q all of the memory modules can be loaded in parallel and the data is directly available no matter which partition (MC group) is chosen. This is achieved by storing in Memory Storage Unit i the data for a task which is to be loaded into the i^{th} logical memory module of the virtual machine of size N/Q , $0 \leq i < N/Q$. Thus, no matter which MC group of N/Q processors is chosen, the data from the i^{th} Memory Storage Unit can be loaded into the i^{th} logical memory module of the virtual machine, for all i , $0 \leq i < N/Q$, simultaneously, i.e., in one parallel block transfer. This same approach can be taken if only $(N/Q)/2^d$ distinct Memory Storage Units are available, $0 \leq d < n-q$, however, 2^d parallel block loads will be required instead of just one. In general, a task needing RN/Q processors, $1 \leq R \leq Q$, logically numbered 0 to $(RN/Q) - 1$, will require R parallel block loads if the data for the memory module whose high-order $n - q$ logical address bits equal i is loaded into Memory Storage Unit i . This is true no matter which group of R MCs (which agree in their low-order $q - r$ address bits) is chosen. If only $(N/Q)/2^d$ distinct Memory Storage Units are available, $0 \leq d \leq n - q$, then $R * 2^d$ parallel block loads will be required instead of just R .

The *Memory Management System (MMS)* controls the loading and unloading of the PCU memory modules. It employs a set of four cooperating, dedicated microprocessors. The *Directory Processor (DP)* receives requests

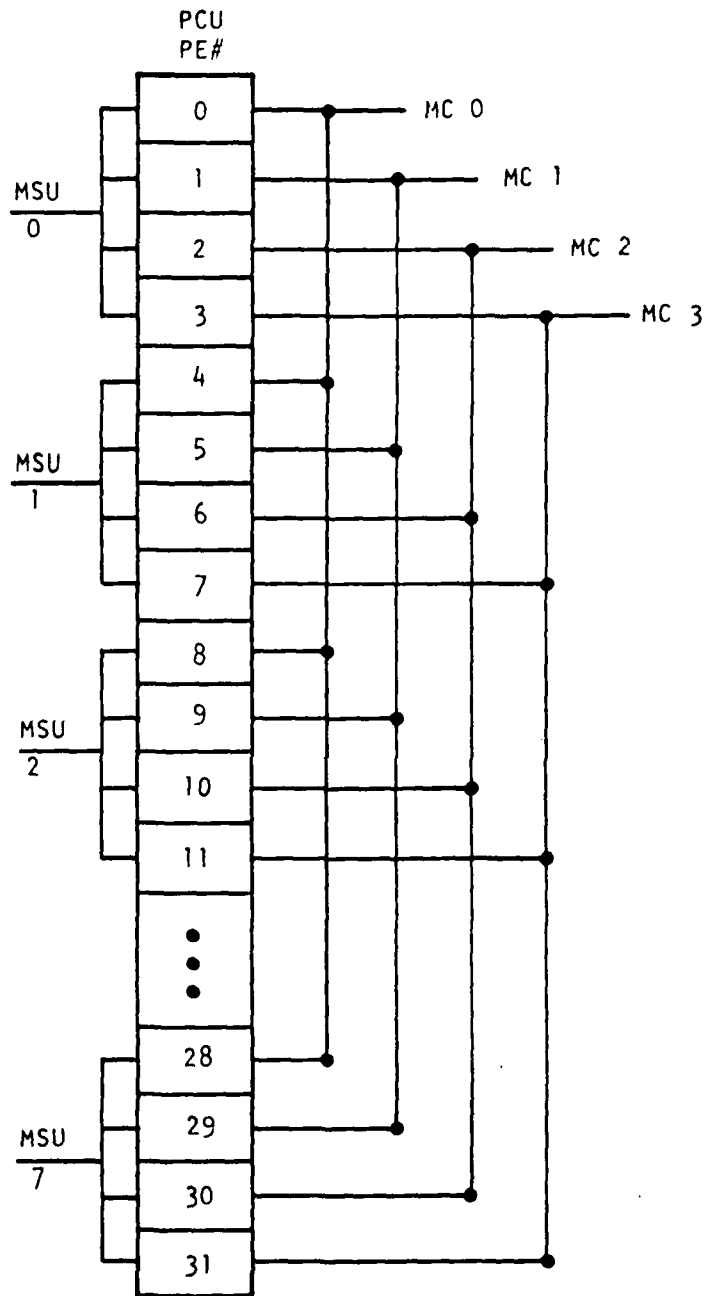


Figure 2.6 Organization of the PASM Memory Storage System for $N=32$ and $Q=4$, where "MSU" is Memory Storage Unit, "MC" is Micro Controller, and "PCU" is Parallel Computation Unit.

from the SCU and the MCs to load or unload PE memory modules. In turn it generates commands for and coordinates the actions of other MMS processors. The *Memory Scheduling Processor (MSP)* receives the commands from the Directory Processor and determines the order in which they should be performed. The *Command Distribution Processor (CDP)* issues these commands to the MSUs and processes command completion acknowledgements. The *Input/Output Processor (IOP)* handles the transfer of files between the MSUs and peripheral devices. It also coordinates the reformatting and distribution of files among the MSUs.

This distributed processing approach is chosen in order to provide the Memory Management System with a large amount of processing power and high speed (due to the parallelism possible) at low cost. A large amount of power is required because it is not desirable to burden the SCU with any memory management tasks. Furthermore, the number of files to be managed is enormous: a user request for an input file for a 1024-PE SIMD program would involve the management of 1024 file directory lookups and transfers. The management problem becomes more severe when multiple simultaneous users of PASM are considered.

The *System Control Unit (SCU)* is a conventional machine, such as a PDP-11/70, and is responsible for the overall coordination of the activities of the Memory Management System and the Micro Controllers. In addition, the SCU is capable of functioning independently as a serial processor. While the rest of PASM executes a parallel computation, the SCU can handle such tasks as program development and job scheduling. In order to perform these functions, the SCU must contain the PASM language compilers and assemblers and portions of the PASM operating system, PASMOS [Tuo83].

Figure 2.7 shows a block diagram of the PASM system prototype that is under construction. The interconnection network used in the prototype, which will be an Extra Stage Cube (see Chapter 3), is not shown in this figure. The prototype will include $N = 16$ PEs and $Q = 4$ MCs. Thus, each MC will control $4 (= N/Q)$ PEs. The System Control Unit will be a dedicated microprocessor. Users will access the PASM prototype system via a Purdue University Engineering Computer Network (ECN) machine as shown in the figure. Commands (jobs) initiated by a user are sent from the ECN machine on which that user is logged to the prototype SCU. Response(s) to a user command (other than large files, such as processed imagery) are returned by the SCU.

For additional information about various aspects of PASM see: organization [SMS78, SSK81], instruction set [SMS78], masking schemes for enabling and disabling PEs [Sie77a, Sie77b, SMS78, SSK81], interconnection networks [AdS82b, AdS82d, Sie77a, Sie79a, Sie79b, Sie80, SiS78], operating system [SSK81, TuS82a, TuS82b, TuS83, TuS84a], programming language [CIS83, MSS80a], memory management system [KSG83, SKW79, SSK81, TuS83, TuS84b], prototype design and simulation [KSH82], and examples of possible applications [KST85, MSS80b, SSE80, SSF82, WaS82]. A reading list of over 50 PASM-related papers is in [SSD84].

2.3 Numerical Aerodynamic Simulation Facility

The *Numerical Aerodynamic Simulation Facility (NASF)* [Bur79] is intended to support execution, in ten minutes or less, of time-averaged Navier-Stokes computations on steady fluid flow problems involving a million grid points. Scientists at the National Aeronautics and Space Administration need this processing power for their research. This requires an average rate of

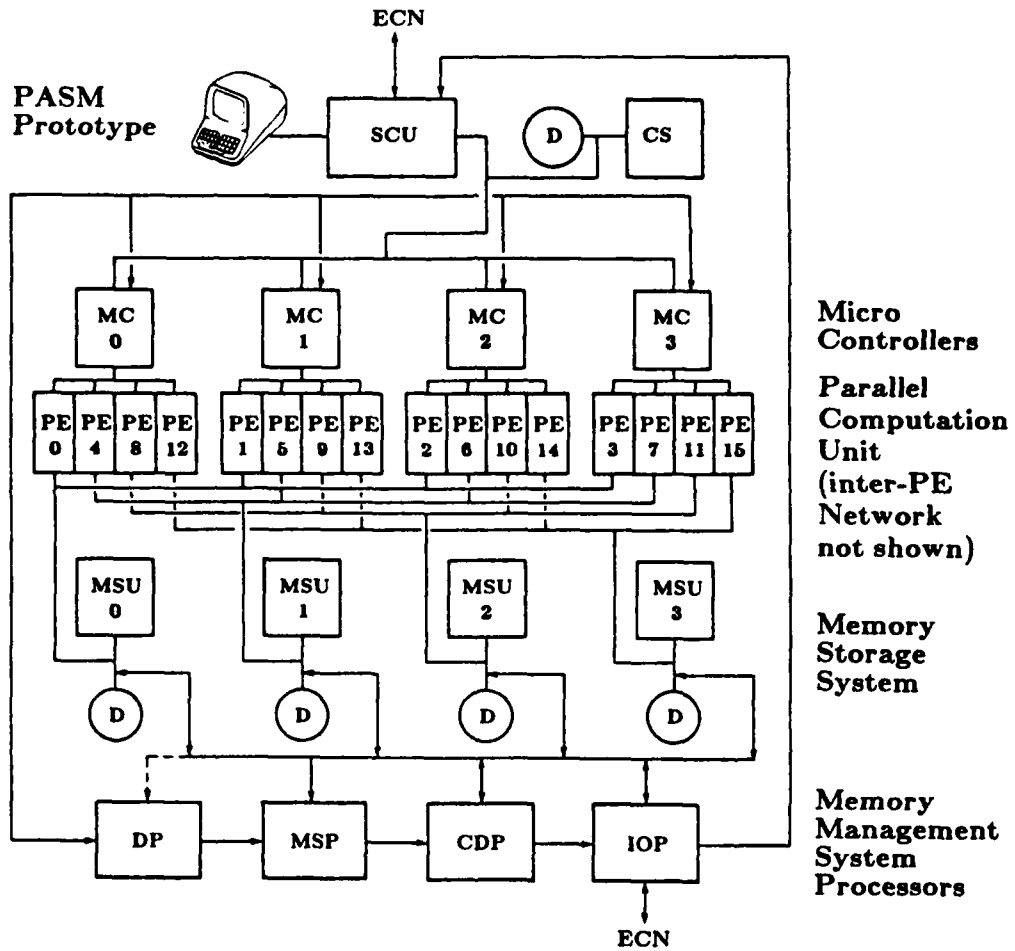


Figure 2.7 Block diagram of proposed PASM prototype.

execution of roughly one billion floating point operations per second. A block diagram of system hardware is shown in Figure 2.8.

The *Flow Model Processor (FMP)* is the core of the system. Figure 2.9 shows the organization of the FMP. The FMP has a structure akin to the P-to-M model shown in Figure 1.2. It has 512 processors (that meet the definition of PE given in Chapter 1); a bidirectional, circuit-switched, multistage interconnection network [BaL81] (called the *Connection Network (CN)*) based on the Omega network [Law75]; 521 *Extended Memory (EM)* modules; a *Data Base Memory (DBM)*; a *Coordinator (CR)*; and a *Diagnostic Controller (DC)*. There are four on-line spares each of processors and EM modules to enhance FMP reliability. The processors each contain a scalar execution unit and storage for data and programs; both integer and floating point arithmetic units are included. The EM contains the data common to the processes being independently evaluated by each of the processors. The DBM is slower than the EM and is provided to hold the past, present, and future jobs scheduled on the system.

The CR has connections to both sides of the network (24 signals each) which allow it to access any memory module or processor. There is a four-bit command bus plus strobe connected to all processors for synchronization and diagnostic purposes. The CR has an I/O channel to the host and generates clock signals for and receives error interrupt signals from the memory modules. It can also issue descriptors to and receive status from the DBM, and there are connections to the DC.

Under normal operating conditions, the switching elements of the CN provide straight and exchange connections bidirectionally. Under special command from the CR, a broadcast connection can be established. A path is

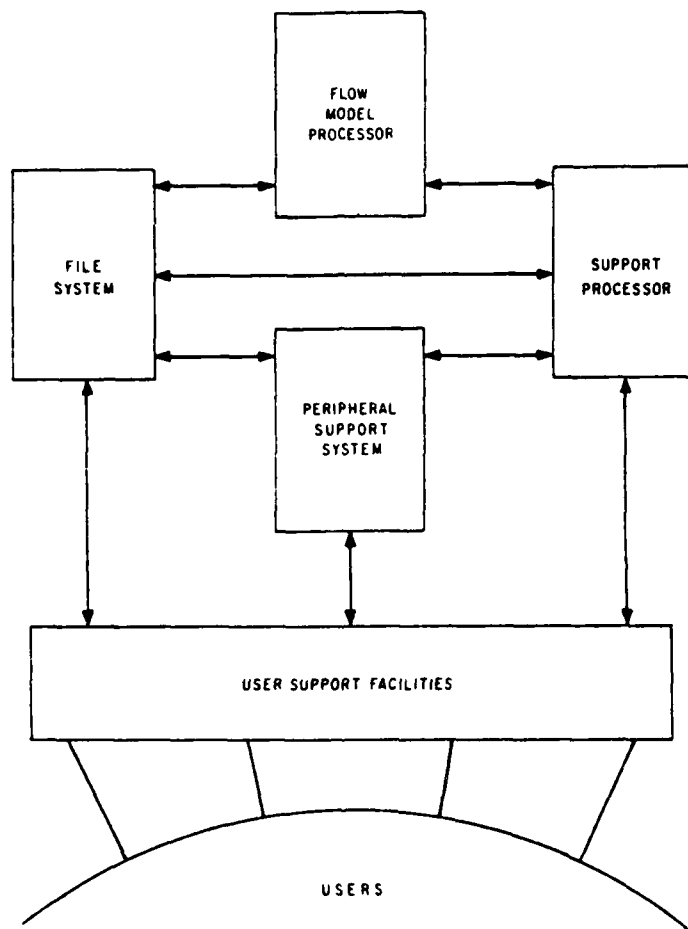


Figure 2.8 Block diagram of the Numerical Aerodynamic Simulation Facility [Bur79].

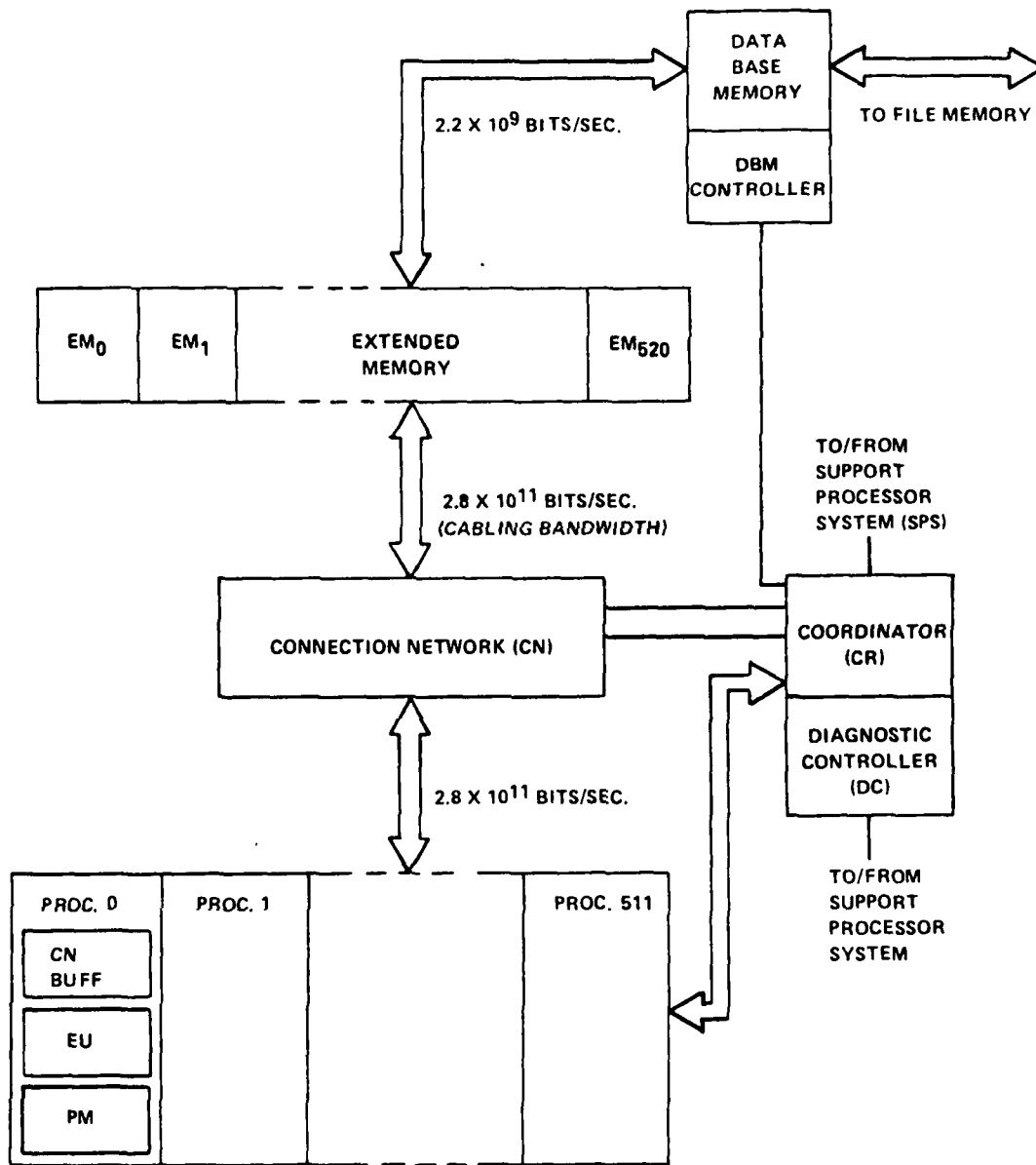


Figure 2.9 General organization of the Flow Model Processor [Bur79].

created from the CR's connection on the memory module side of the network to all processors. The CR can also specify that an arbitrary module broadcast a specified data item to all processors.

Another special connection that can be initiated by the CR is a direct exchange of data by pairs of processors. This "wraparound" connection allows all pairs of processors whose addresses differ in the i -th bit to swap data items.

The CR is also responsible for program initiation and diagnostics. It has access via the network to every processor register and, generally, can exercise any intraprocessor activity. Program loading, transfers between disk and DBM, and some diagnostics are handled by the host computer. The DC is an interface between the host computer and the CR; it allows the host to diagnose the CR.

All connections established in the CN are strictly circuit switched. Two options were, however, considered for establishing circuits. The first method provides two registers in each node for latching routing tag and memory address information. This makes establishing a circuit very fast since the interface does not have to wait until the circuit is complete before transmitting the memory address. The second method, tentatively chosen because of its lower cost, includes no such registers. The interface must hold the routing tag on the data path until the circuit is complete. Due to the anticipated use of the system in which all processors participate in computations, there is no attempt to take advantage of the partitioning abilities of the network [Sie80]. Partitionability is considered only to distribute sections of the network advantageously among equipment cabinets.

The *Support Processing System* serves as the central control, interfaces with users and peripherals, maintains data files, and provides computational support necessary to keep FMP utilization high. It consists of three subsystems: the Support Processor, the File System, and the Peripheral Support System (see Figure 2.8).

The *Support Processor*, which acts as a system host processor, is a dual-processor Burroughs B7800. The planned configuration includes redundancy in most elements of the B7800 for high availability. Since the Support Processor is the master control for the NASF facility, most user communication is supported by it. Up to 96 input lines are envisioned.

Study of output device requirements indicated that some anticipated peripherals would need a significant amount of computational support. The Computer Output to Microfilm (COM) device is the most demanding of these. NASA postulates a COM load in excess of 10,000 frames per day, with output assumed to be graphic images. The majority of this load is for "movies" of simulation results. The *Peripheral Support System* is configured with two high-end minicomputers with special-purpose software for handling these tasks.

The volume of data and programs to be moved in and out of the FMP together with the amount of file management required for the total system indicate strongly that a separate system be provided for this purpose, rather than using the Support Processor (B7800) itself. Also, with file management functions in a processor separate from processors executing user programs, security capability is enhanced. The *File System* includes the disk packs, the archival store, and the file manager. It performs directory management and storage allocation functions and is responsible for data ownership and access controls. The DBM may also be considered part of the File System because it

is a staging area for FMP programs and data.

Achieving adequate fault tolerance is an important design issue for NASF. Many features are built into the system to provide fault tolerance. First, each 48 bit data word is accompanied by a 7-bit single-error-correcting/double-error-detecting (SECDED) code throughout the system. To maintain the long term integrity of the data stored in the DBM, each data item is periodically "scrubbed." That is, all single bit errors are corrected often enough to reduce drastically the chance of a double error. Second, whenever a memory request is made, the selected memory module compares the incoming routing tag to its own address, detecting misrouted packets. Any discrepancy is reported as an error by sending an interrupt to the CR. Third, there are four spare processor-memory pairs and four spare memory modules, all on-line. If diagnostic tests detect a faulty processor, for example, a new processor-memory pair can be automatically switched into its place. The switch is accomplished by performing an address transformation in hardware. Finally, automatic recovery of the file system is to be implemented in software. All errors of any kind are recorded and reported.

Effort has been made toward developing a language for NASF called FMP FORTRAN. FMP FORTRAN is based on American National Standards Institute (ANSI) FORTRAN 77 [ANS78] with extensions and modifications to improve its utility for the planned applications and to allow more efficient use of FMP hardware. The additional language constructs provide means of describing both spatial (geometry and state) and temporal (processes) relationships in a simulation model. The additional language constructs are as follows.

1. DOMAIN: Used to define all those discrete points in the simulation model grid at which state information will exist and/or processing will occur (similar to array dimensioning).
2. REGION: Allows creation of a virtual domain of interest through dynamic selection of elements of a DOMAIN.
3. INALL: Provides for declaration of variables associated with grid points in a simulation model without using subscripts.
4. DOALL ... USING
...
ENDDO ... GIVING: Used to express the inherent concurrency in a program loop. The USING and GIVING portions of the construct explicitly define data dependency of the simulation model so that the compiler can anticipate requests for data transfer between the EM and processors.
5. SUMALL: Instructs a processor to sum all instances of an indicated variable stored in that processor's associated memory.
6. LOCATION: Returns the instance number of the successful instance of the most recent execution of MAXALL (or MINALL), which finds the maximum (or minimum) value in a data structure.
7. RECURRENCE: Uses recursive doubling to solve recurrence relations on one-dimensional domains.

During normal NASF operation, all data and program code for a task is first loaded into the DBM. DBM loading is scheduled by the Support Processor via the File System. The Support Processor scheduler initiates a task on the FMP through interaction with the *coordinator (CR)*. When task execution is

to begin, software in the CR starts transfer of code files from the DBM to the EM. From there, the CR causes its code files to be loaded into its memory and causes processor code files to be broadcast to each processor. The initialization phase of the program (now in the CR) then transfers necessary data to the EM. With data in place in the EM and code files in place in the CR and processors, task execution starts. While task execution is in progress, the CR serves as a high-level "instruction sequencer"; processor tasks are explicitly initiated, and when all processors complete their tasks the CR initiates the next task.

2.4 Texas Reconfigurable Array Computer

The *Texas Reconfigurable Array Computer (TRAC)* is a microprocessor-based partitionable SIMD/MIMD system [LiT77, SUK80]. A prototype has been built at the University of Texas at Austin. The important attributes of the design include memory space sharing, reconfigurability, varistructuring, inter-task communication ability, and its appearance as a virtual machine to the user. Memory space sharing means independent or interacting tasks can run simultaneously. TRAC can dynamically change existing partitions of processors and other resources, making it reconfigurable. Varistructuring refers to the fact that processors can be grouped to process various data widths. The system is virtual in that programs need not know on what set of processors they are executing. Figure 2.10 gives a block diagram of TRAC.

TRAC processors operate on eight-bit data, but can be linked together for multi-byte operations [KPL80]. Each contains a microcoded *Register Arithmetic Logic Unit (RALU)*, several control units, a packet buffer, control storage and a sequencer, and status registers. The RALU uses two 2901 bit-sliced, microprogrammed processors. The status registers store substantial

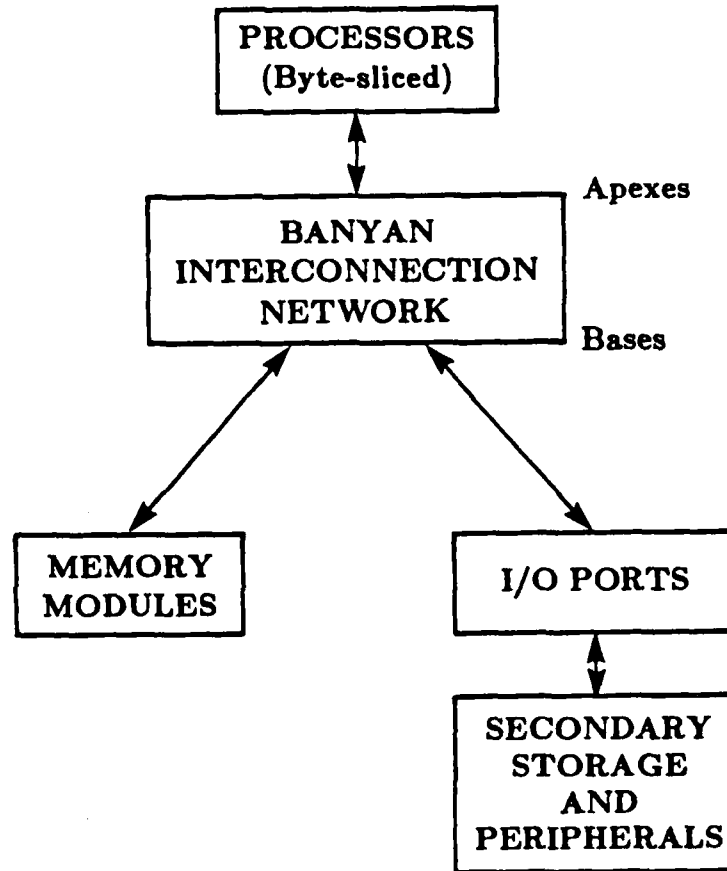


Figure 2.10 Block diagram overview of TRAC.

information concerning system operational and structural condition.

An *SW-banyan* multistage, bidirectional interconnection network links the processors to the memory modules and I/O ports (MIOs) [PKM80]. The network consists of apex, base, and intermediate nodes. Processors are attached to the apex nodes, while MIOs are connected to base nodes. The intermediate nodes represent switching elements. The network as implemented is capable of three tree-structured types of connections: data trees, instruction trees, and shared memory trees. A *data tree* links a single processor to several MIOs for *Single Instruction Stream - Single Data Stream (SISD)* mode operation. An *instruction tree* connects several processors to a base node of the network, making SIMD execution possible. A *shared memory tree* is similar to an instruction tree except that any one of the processors on a shared tree can exclusively acquire the shared MIO of the tree. An explicit release of the MIO is required before other processors of the tree have the opportunity to establish control of the MIO. There is also a controller for the network used to create the trees.

Embedded in any instruction or shared memory tree is a *General Purpose Communication (GPC)* link. The GPC link provides a high speed, bit serial, bidirectional communication path among the group of processors associated with an instruction or shared memory tree. For instruction trees the GPC paths might be used for carry propagation in high precision arithmetic. Also, conflicting memory requests in shared memory trees can be arbitrated through use of GPC links.

Data transfer in the trees is via circuit switching. However, the network also supports packet switching in the background. The MIOs are synchronized, so during the memory access cycle the links of the network are idle and, thus,

free to forward packets. Packets are sent by explicit instruction or as an implicit part of some processor microinstruction.

Secondary storage, called the *Self-Managing Secondary Memory (SMSM)*, is attached to one I/O port. It is capable of storing variable length records along with a label; SMSM hardware can search for records by their labels. Other I/O ports can support various peripherals including terminals.

2.5 STARAN

STARAN is a system built by Goodyear Aerospace Corporation, designed to combine the attributes of parallelism, associativity, and low cost [Bat74]. An *associative* processor is an SIMD machine in which data can be retrieved using the associative concept, i.e., by contents rather than by physical location [Bae80]. Low cost was achieved by using off-the-shelf components. A more recent version, the STARAN Series E, is essentially the same system, but with more memory and implemented with some higher speed circuitry [Bat77b]. Applications envisioned for STARAN by the designers included Fast Fourier Transform (FFT) operations, sonar post-processing involving FFTs, high-speed character string searching, file (data base) processing, and air traffic control. Figure 2.11 shows a block diagram of a typical STARAN system.

A host computer is not required for STARAN operation. The port is provided so that STARAN can be operated as a special-purpose peripheral with respect to a host. STARAN hosts have included a Honeywell HIS-645 connected via an I/O channel and an XDS $\Sigma 5$ computer interfaced through a direct memory access port.

The *AP Control* exercises control over its attached *Array Modules*; all arrays follow the same instruction stream. It fetches instructions from the *Control Memory*, which holds the AP Control programs, PIO control programs, and microprogram subroutines. AP Control includes registers to hold the current instruction; a program status word; a comparand word, operand for the array modules, or result from the array modules; the active/inactive select for each array module; four pointers for use in array module memory access; three loop counters; and two array module memory access mode words.

The *Parallel Input/Output (PIO)* is used for high bandwidth I/O and data transfers between array modules. The *PIO Control Unit* controls the *PIO Flip Network* and the attached array modules. It receives data and instructions from the control memory. The PIO flip network switches data among eight 256-bit ports. Ports 0 through 3 connect to the four array modules. Port 7 connects to a bus in the PIO Control. Ports 4, 5, and 6 are spares and can be used for high bandwidth peripherals or additional array modules. The PIO Control Unit transfers data to and from array modules not being used by AP Control.

A Digital Equipment Corporation (DEC) PDP-11 minicomputer is used for the *Sequential Control*, performing peripheral handling, system console, and diagnostic functions. Sequential Control can access any part of control memory. Synchronization of Sequential Control, AP Control, and PIO Control is accomplished by the *External Function (EFX)* logic. Control units issue commands to EFX logic to cause system actions and read system states.

The array modules perform the data processing for STARAN. Figure 2.12 is a diagram of an array module. A key component of an array module is the *multidimensional access (MDA)* memory. The MDA memory

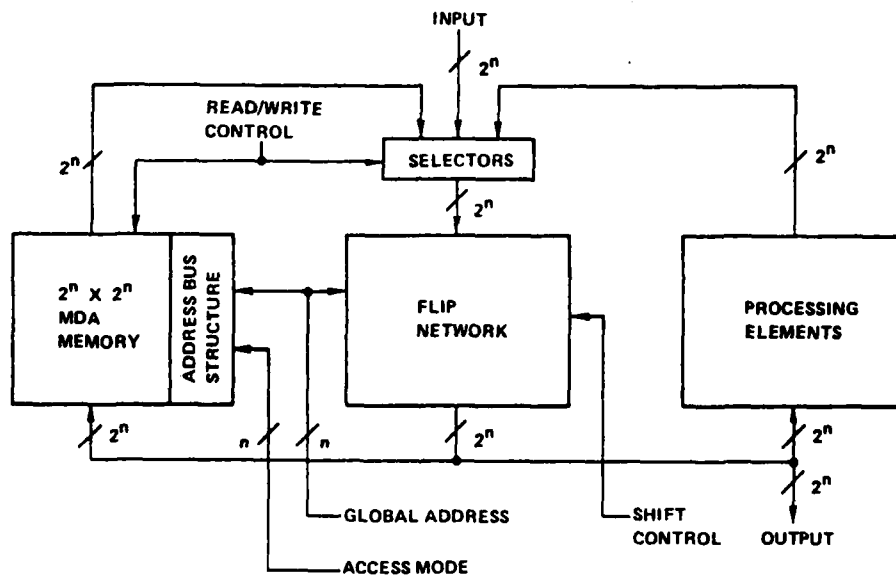


Figure 2.12 STARAN array module [Bat76].

stores $2^{16} = 64\text{K}$ bits of information, implemented as 256 words of 256 bits. In STARAN Series E, an enhanced version of STARAN, the maximum MDA storage in each array module was expanded to $2^{24} = 16\text{M}$ bits [Bat77b]. Reading and writing is controlled by specifying the shape and position of an access stencil [Bat77a]. An eight-bit access mode word, M , specifies stencil shape, and G , an eight-bit global address word, determines stencil position. Every stencil refers to exactly $2^8 = 256$ bits of storage. The specified bits can be fetched or stored in one memory cycle.

A flip network is also used in each array module to provide communications between the MDA memory and the 256 one-bit processing elements of the module, and between processing elements. The network operates in circuit switched mode. The *Selector* determines the parties to an information transaction. Given the structure formed with the aid of the Selector, STARAN can be considered both a PE-to-PE and P-to-M parallel processor (even though the network is unidirectional).

In an array module there are three 256-bit registers (M , X , and Y) that perform bit-oriented operations, one bit associated with each processor. Register M drives the write mask bus of the MDA memory to select which of the MDA memory bits are modified in a masked-write operation. Registers X and Y are used for data processing. If f_i is the i^{th} output of the flip network, x_i and y_i are the i^{th} bits of X and Y , respectively, and ϕ denotes any Boolean function, then an array module can perform the following operations.

1. $x_i \leftarrow \phi(x_i, f_i)$.
2. $y_i \leftarrow \phi(y_i, f_i)$.

3. Operations (1) and (2) simultaneously using the same Boolean function.
4. Operation (1) only when $y_i = 1$, else x_i unchanged.
5. Operation (4) with the current y_i , plus operation (2) simultaneously producing a new y_i .

There is a *resolver* connected to the Y register. The resolver reads the state of the Y register, and if any bit of Y is set, the *activity-or* output of the resolver is set. If some Y bits are set, the address output lines of the resolver indicate the bit position of the first set bit. The result of an associative search of the MDA is stored in the Y register. The resolver thus provides a SOME/NONE match flag and information to implement the SELECT FIRST RESPONDER function [Fos76].

2.6 PUMPS

PUMPS is a systems being designed at Purdue University for pattern and image analysis [BFH82]. An underlying concept used in designing PUMPS is that cost-effective solutions to many applications such as pattern analysis require some form of functional specialization in the computer architecture. Figure 2.13 depicts the architecture of PUMPS.

Functional specialization is expressed in PUMPS on a system level by its three major subsystems: man-machine interface, image analysis, and data base management. There are P *task processing units (TPUs)*, each of which is multiprogrammed. They can operate in an interactive fashion through the shared *peripheral processors and VLSI units (PPVUs)* and *shared memory (SM)* system. The *special resource arbitration network (SRAN)* can link a TPU to a chosen PPVU or provide a path between PPVUs. It is some low

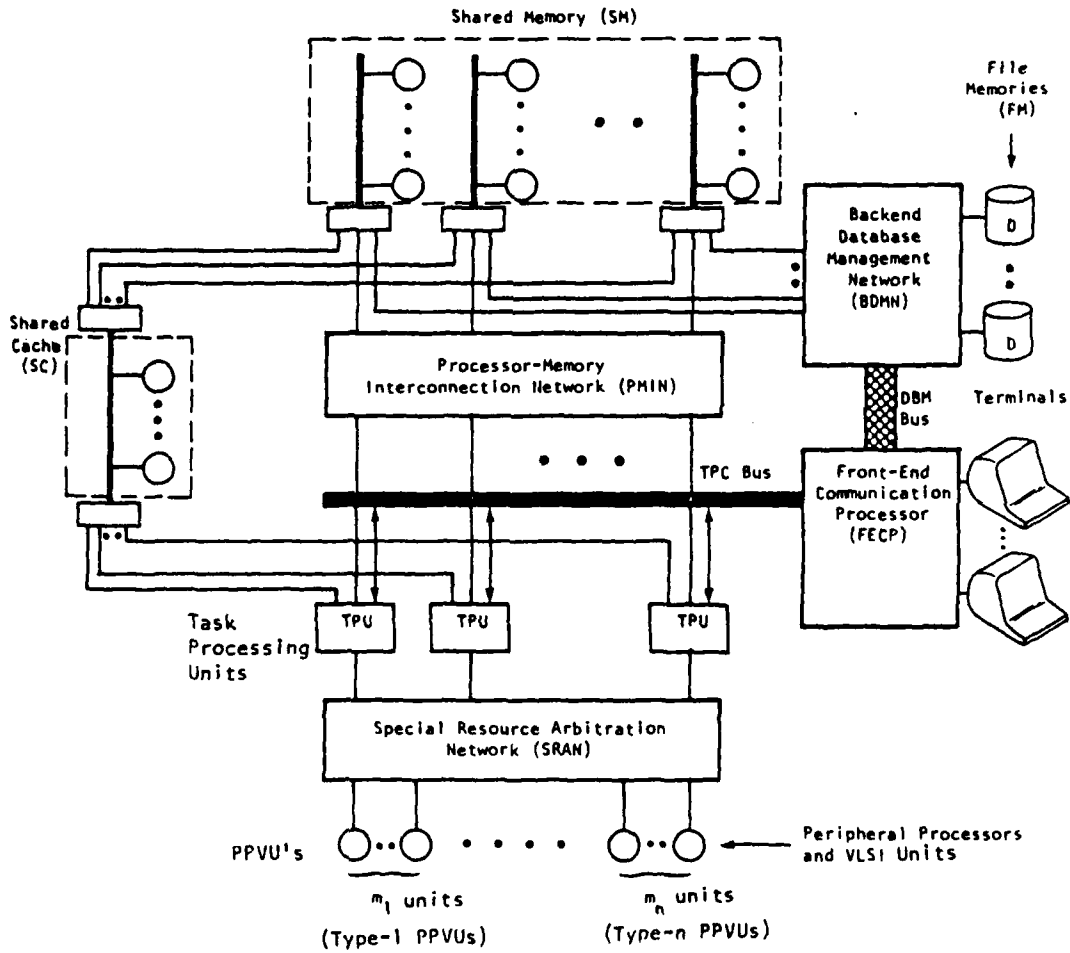


Figure 2.13 System architecture of PUMPS [BFH82].

conflict multistage interconnection network. Allocation of PPVUs to the TPUs is dynamic and depends on the needs of the currently executing task. The specific functions provided by the PPVUs can be varied to suit application requirements. Thus, the PPVUs realize functional specialization in PUMPS on a more local level. The PPVUs together are the heart of the image analysis subsystem of PUMPS. The TPUs pass information among themselves using the *task processor communications (TPC)* bus. Such communication is envisioned as consisting mainly of interrupts, synchronization signals, and control signals. TPCs initiate PPVU tasks, execute purely sequential tasks, and participate in both MIMD and operating system processes.

Within a TPU there is a *task processor (TP)*, a *task cache (TC)*, a *task memory (TM)*, a *task memory management unit (TMMU)*, and a *resource controller with data channels (RCDC)*. The TP, TC, TM, and TMMU together form a conventional serial processor. When there is a TP data request resulting in a TC miss, the TMMU fetches the data from the TM if it resides there. If a TM page fault results, the block-transfer oriented *processor-memory interconnection network (PMIN)* links the TPU with the shared memory to service the fault.

The shared memory is connected to the *file memories (FM)* via the *backend image database management network (BDMN)*. This network is designed to handle data transfer from the multiple disks comprising the FM. The database subsystem of PUMPS, mentioned earlier, consists of the FM, BDMN, and a backend computer (not shown in Figure 2.13) connected to the BDMN.

The *front-end communications processor (FECP)* provides the man-machine interface of PUMPS. It performs such tasks as editing and terminal I/O.

2.7 Conclusions

This chapter has overviewed the PASM, NASF, TRAC, STARAN, and PUMPS parallel processing systems. These machines utilize multistage interconnection networks and, so, could potentially employ the fault-tolerant network, or concepts from it, that is defined and studied in chapters that follow. Further, these systems illustrate that the specific applications to be performed can profitably be taken into account when specifying a parallel computer architecture. With this motivation, a subsequent chapter investigates an image processing task and uncovers what it would require of a parallel computer architecture and its interconnection network.

CHAPTER 3

DEFINITION OF THE EXTRA STAGE CUBE INTERCONNECTION NETWORK

3.1 Introduction

The *Extra Stage Cube (ESC)* [AdS82a, AdS82c, AdS82d, AdS82e, AdS84a] is a fault-tolerant interconnection network intended for use in large-scale SIMD, MSIMD, MIMD, and partitionable SIMD/MIMD computer systems. The ESC is derived from the Generalized Cube interconnection network [SiS78, SiM81b]. It consists of a Generalized Cube network with one additional stage of switching elements at the input and additional hardware to allow the bypass, when desired, of this extra stage or the output stage.

Multistage cube-type networks such as the baseline [WuF80], delta [Pat81], Generalized Cube [SiS78], indirect binary n -cube [Pea77], omega [Law75], STARAN flip [Bat76], and SW-banyan ($S=F=2$, $L=n$) [GoL73] have been proposed for use in parallel computer systems. These include PASM [SSK81], the Flow Model Processor of the Numerical Aerodynamic Simulator [Bur79], TRAC [LiT77, SUK80], STARAN [Bat74], PUMPS [BFH82], the Ballistic Missile Defense Agency distributed processing test bed [McW78], Ultracomputer [GGK83], and data flow machines [DBL80]. The ESC can be used in any of these systems to provide fault tolerance in addition to the usual cube-type network communication capability.

The ESC can be used in various ways in different computer systems. For example, consider how the ESC could be incorporated in the PASM and PUMPS systems. In the context of PASM (see Figure 2.2), the network would operate in a unidirectional, PE-to-PE packet-switched mode [SiM81b]. The PUMPS MIMD architecture (see Figure 2.13) [BFH82] consists of multiple processors with local memories which share special purpose peripheral processors, VLSI functional units, and a common main memory. The network would serve in a bidirectional, circuit-switched environment for this architecture, connecting local memories to the common main memory.

3.2 Definition of the Generalized Cube Network

The Generalized Cube network is a multistage cube-type network topology which was presented in [SiS78]. This network has N input ports and N output ports, where $N = 2^n$. Figure 3.1 shows it for $N = 8$. The network ports are numbered from 0 to $N-1$. Input and output ports are network interfaces to external devices, called sources and destinations, respectively, and have addresses corresponding to their port numbers. The Generalized Cube topology has $n = \log_2 N$ stages, where each stage consists of a set of N links connected to $N/2$ switches. Thus, the Generalized Cube has $O(N \log N)$ physical components. Note that neither a network input port nor output port is considered a link.

Each switch is an *interchange box*, a 2-input/2-output device, and is individually controlled. An interchange box can be set to one of four states. Assign the label i to the upper input and output ports of a box, and the label j to the lower input and output ports. The four functional states are: 1) *straight* - input i to output i , input j to output j ; 2) *exchange* - input i to output j ,

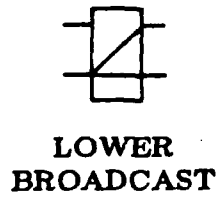
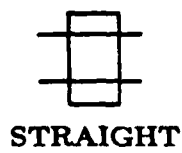
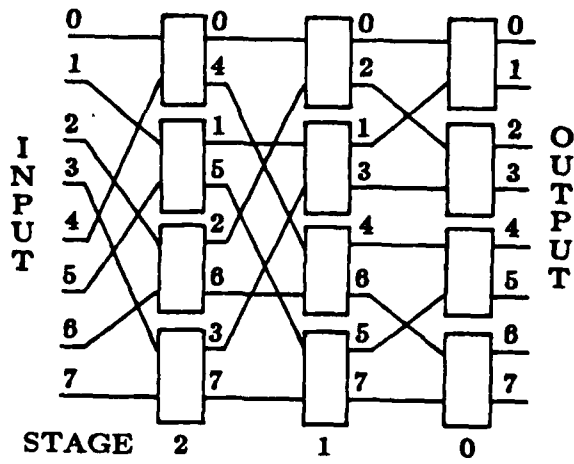


Figure 3.1 The Generalized Cube network with $N = 8$, and the four states of an interchange box.

input j to output i ; 3) *lower broadcast* - input j to outputs i and j ; and 4) *upper broadcast* - input i to outputs i and j [Law75]. Figure 3.1 shows these four states.

All Generalized Cube links and interchange boxes are assigned labels. The input ports of the boxes of the input stage are labeled by the number of the network input port to which they are connected. In all stages the upper output of a box has the same label as the upper input of that box; the same relationship holds true for the lower box input and output labels. Links take the label of the box output to which they are connected. Links give their label to the box inputs to which they attach. This is shown in Figure 3.1.

An interconnection network in an SIMD environment can be described as a set of *interconnection functions*, where each is a permutation (bijection) on the set of PE addresses, or network input/output (I/O) port labels [Sie77a]. When interconnection function f is applied, network input S is connected to output $f(S) = D$ for all S , $0 \leq S < N$, simultaneously. That is, saying that the interconnection function *maps* the source address S to the destination address D is equivalent to saying the interconnection function causes data sent on the input port with address S to be *routed* to the output port with address D . SIMD systems typically route data simultaneously from each network input via a sequence of interconnection functions to each output. For MIMD systems, communication from one source is typically independent of other sources. In this situation the interconnection function is viewed as being applied to the single source, rather than all sources.

The connections in the Generalized Cube are based on the cube interconnection functions [Sie77a]. Let $p_{n-1} \dots p_1 p_0$ be the binary representation

of P , $0 \leq P < N$. Then the n cube interconnection functions are defined as

$$\text{cube}_i(p_{n-1} \dots p_1 p_0) = p_{n-1} \dots p_{i+1} \bar{p}_i p_{i-1} \dots p_1 p_0$$

$0 \leq i < n$, $0 \leq P < N$, and \bar{p}_i denotes the complement of p_i . This means that the cube interconnection function maps P to $\text{cube}_i(P)$, where $\text{cube}_i(P)$ is a label that differs from the label P in just the i^{th} bit position. Stage i of the Generalized Cube topology contains the cube_i interconnection function. The input labels of each stage i box differ in the i^{th} bit position, hence, the exchange setting will implement cube_i in mapping the box input labels to the box outputs. Thus, data items input to that interchange box are transferred as specified by the cube_i interconnection function. When set to straight, data items input are transferred according to the identity function, where $\text{identity}(p_{n-1} \dots p_0) = p_{n-1} \dots p_0$. Since each interchange box is individually controlled, each stage i may perform the cube_i interconnection function on some subset of the data items depending on the settings of the interchange boxes. Stage i is the only stage which can map a source to a destination with an address different from the source address in the i^{th} bit position.

This network is called the Generalized Cube by virtue of a geometric relationship that can be established among the network port addresses. Network port addresses can be assigned to the corners of an n -dimensional cube in such a way that all address pairs of the form $\{P, \text{cube}_i(P)\}$ are on adjacent corners. Further, for a given i all edges defined by $\{P, \text{cube}_i(P)\}$ are parallel. Data transfer in the Generalized Cube can be visualized as movement along hypercube edges in a fixed sequence of dimensions. Figure 3.2 shows a three-dimensional cube with the vertices labeled in base 2 to correspond to the Generalized Cube for $N=8$. The edges of the cube correspond to the three cube functions for a network of this size. Vertically oriented edges correspond

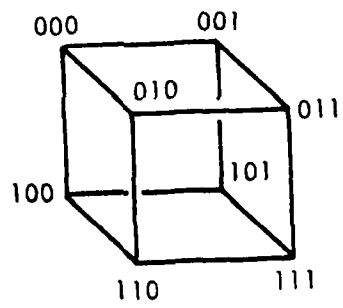


Figure 3.2 Relationship of the Generalized Cube network to an n -dimensional cube for $N = 8$, i.e., $n = 3$.

to the cube_2 function, as they connect vertices with labels differing in the high order, or second, bit position. Diagonal edges represent the cube_1 function, and horizontal edges the cube_0 function.

There is a class of cube-type networks of which the Generalized Cube is representative. By combining the results of [Sie79a, SiS78, FeW81, WuF80] it can be seen that all of the following networks are topologically equivalent: the Generalized Cube [SiS78], the STARAN flip network [Bat76], the omega network [Law75], and the indirect binary n-cube network [Pea77]. (The SW-banyan ($S=F=2$, $L=n$) is defined as a graph [GoL73] and has the same topology as a multistage cube [WuF80].) For this reason the Generalized Cube can be used as a standard for comparing cube-type networks with other interconnection networks. In this case the comparison will be with the Extra Stage Cube.

More information on the Generalized Cube can be found in [SiM81b, Sie85]. Implementation details are discussed in [McS80].

3.3 Properties of the Generalized Cube Network

The Generalized Cube has been shown to have many useful properties, summarized below. This listing is provided as background for considering the ESC network. The ESC has all of the properties of the Generalized Cube network in addition to certain other capabilities discussed in Chapter 4.

The Generalized Cube can

1. handle up to N simultaneous transfers of information;
2. be controlled in a distributed fashion using routing tags;

3. be partitioned into independent subnetworks;
4. perform broadcasting of information from one device to all or a subset of devices using the network;
5. function in SIMD, MSIMD, MIMD, and partitionable SIMD/MIMD modes of parallel processing; and
6. be implemented in numerous ways.

3.4 Definition of the Extra Stage Cube Network

The ESC is formed from the Generalized Cube by adding an extra stage along with a number of multiplexers and demultiplexers. Figure 3.3 illustrates ESC network structure for $N=8$. The extra stage, stage n , is placed on the input side of the network and implements the cube_0 interconnection function. Thus, there are two stages in the ESC which can perform cube_0 . The incremental increase in hardware complexity for the ESC relative to the Generalized Cube is $O(N)$; overall ESC hardware complexity remains $O(N \log N)$. Thus, it has relatively low incremental cost over the Generalized Cube network.

For the ESC, as for the Generalized Cube, a stage is considered to consist of a bank of interchange boxes and the links from their outputs. Thus, the links connecting stage i boxes to stage $i-1$ boxes are stage i links. This definition is used for the consistent treatment it allows of both boxes and links with respect to the fault tolerance discussion of Chapter 4. The supporting reason for this decision is given in Chapter 4, Section 4.3.3. Note that there are no stage 0 links under this definition (recall that network ports are not considered links).

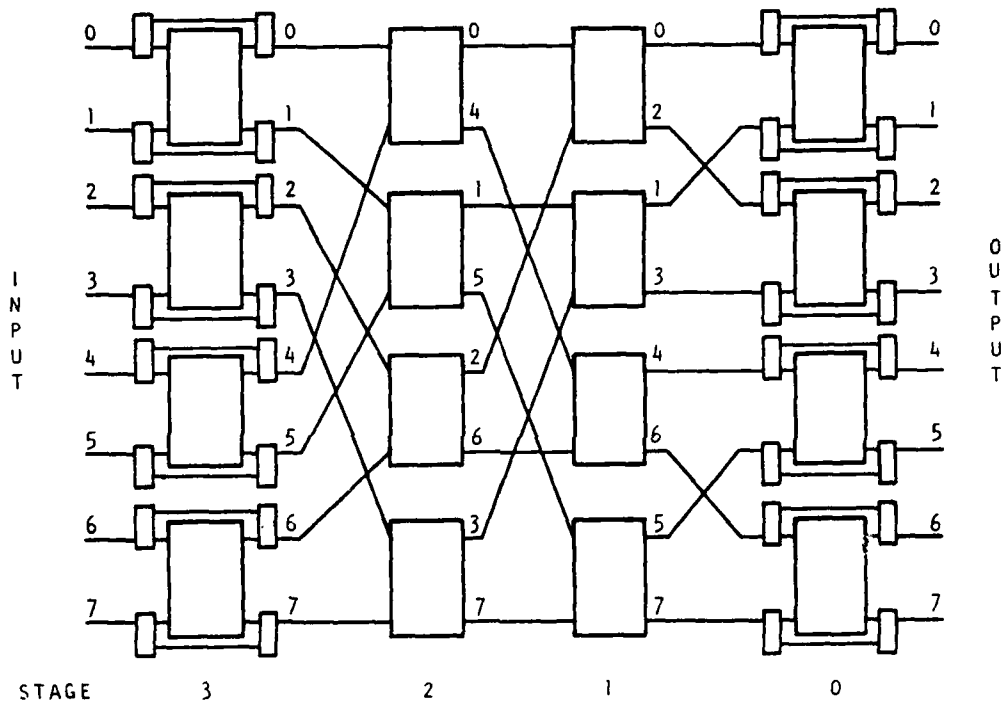


Figure 3.3 The Extra Stage Cube network with $N = 8$.

Stage n and stage 0 can each be enabled or disabled (bypassed). A stage is *enabled* when its interchange boxes are being used to provide interconnection. It is *disabled* when its interchange boxes are being bypassed. Enabling and disabling in stages n and 0 is accomplished with a demultiplexer at each box input and a multiplexer at each output. Figure 3.4 details an interchange box from stage n or 0. One demultiplexer output goes to a box input, the other to an input of its corresponding multiplexer. The remaining multiplexer input is from the matching box output. The demultiplexer and multiplexer are configured such that they either both connect to their box (enable) or both shunt it (disable). All demultiplexers and multiplexers for stage n share a common control signal; those in stage 0 also share a common control signal.

The ESC uses stage n multiplexers to prevent stage n box output failures from affecting stage n links. The multiplexers provide the links with isolation from box output stuck-logic-level faults. The stage 0 input demultiplexers serve the same function for stage 1 links. Preventing these box faults from affecting associated links is important to the fault tolerance capabilities of the ESC, which are discussed in Chapter 4.

Stage n and 0 enabling and disabling are performed by a system control unit. Normally, the network will be set so that stage n is disabled and stage 0 is enabled. The resulting structure is that of the Generalized Cube. Figure 3.5 shows stage n disabled, stage 0 enabled, and the path from input 2 to output 1 under this circumstance for an ESC with $N=8$. If after running fault detection and location tests a fault is found, the network is reconfigured. A fault in a stage n box requires no change in network configuration; stage n remains disabled. If the fault is in stage 0 then stage n is enabled and stage 0 is disabled. Figure 3.6 shows an ESC with $N=8$ and stage n enabled, stage 0

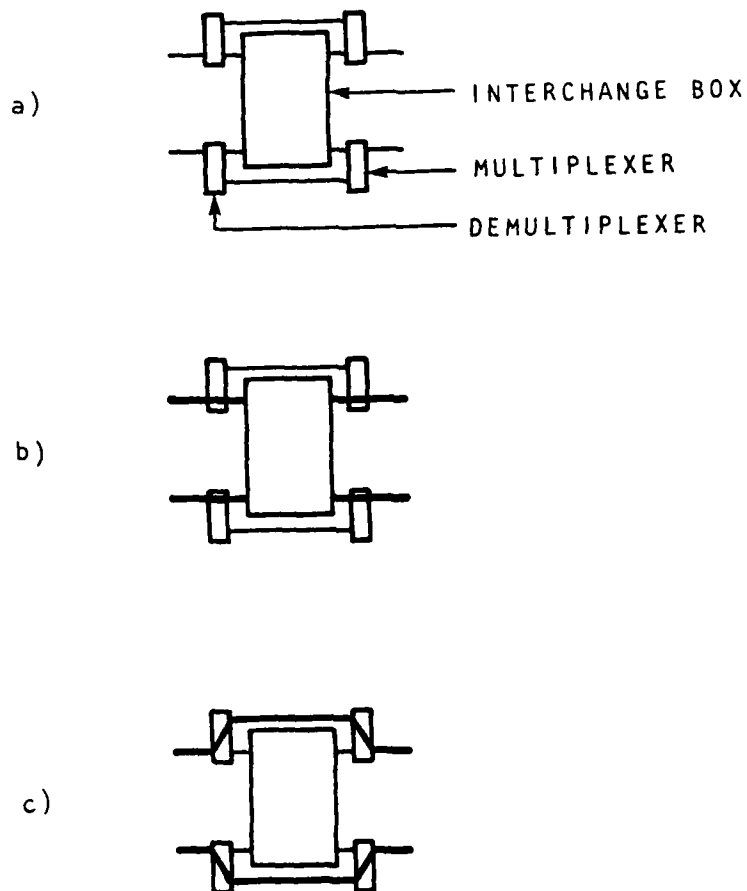


Figure 3.4 (a) Detail of interchange box with multiplexer and demultiplexer for enabling and disabling. (b) Interchange box enabled. (c) Interchange box disabled.

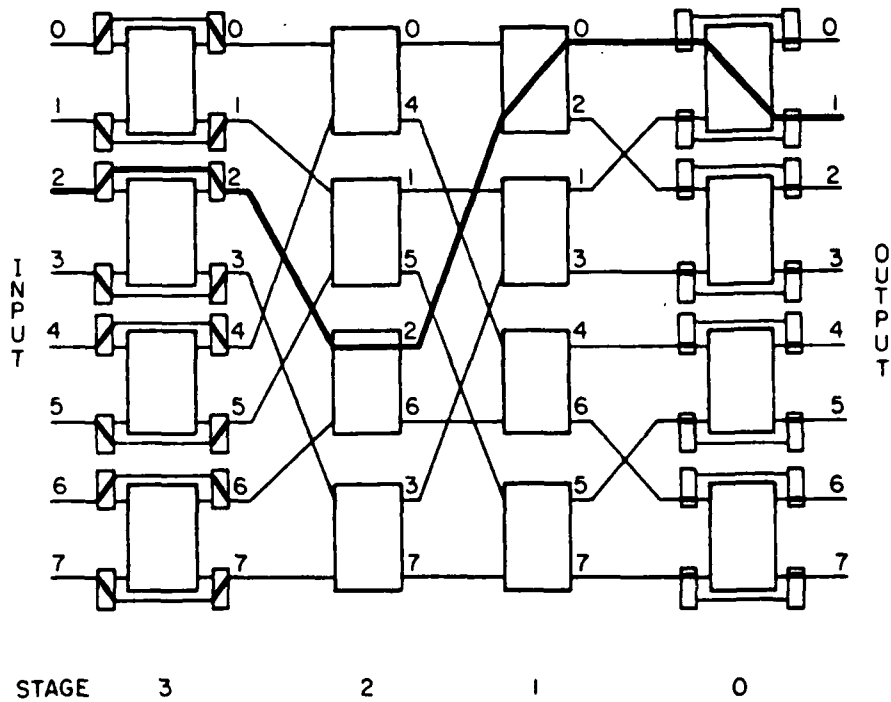


Figure 3.5 The path from input 2 to output 1 in the ESC for $N=8$, when stage n is disabled and stage 0 is enabled. Stage n and 0 multiplexer and demultiplexer settings are shown explicitly [Sie85].

disabled, and the path from input 2 to output 1 in this case. For a fault in a link or in a box in stages $n-1$ to 1, both stages n and 0 will be enabled. Enabling both stages n and 0 provides tolerance to this type of fault by providing two paths between any source and destination, only one of which can contain the existing fault. This is discussed in detail in Chapter 4.

Intuitively, for both the Generalized Cube and the ESC, stage i , $0 < i < n$, determines the i^{th} bit of the address of the output port to which the data is sent. Consider the route from source $S = s_{n-1} \dots s_1 s_0$ to destination $D = d_{n-1} \dots d_1 d_0$. If the route passes through stage i using the straight connection then the i^{th} bit of the source and destination addresses will be the same, i.e., $d_i = s_i$. If the exchange setting is used, the i^{th} bits will be complementary, i.e., $d_i = \bar{s}_i$. In the Generalized Cube, stage 0 determines the 0th bit position of the destination in a similar fashion. In the ESC, however, both stage n and stage 0 can affect the 0th bit of the output address. Using the straight connection in stage n performs routings as they occur in the Generalized Cube. The exchange setting makes available an alternate route not present in the Generalized Cube. In particular, the route enters stage $n-1$ at label $s_{n-1} \dots s_1 \bar{s}_0$, instead of $s_{n-1} \dots s_1 s_0$.

For convenience in later discussion, the term *network component* is defined to be any element comprising the structure of the ESC. Thus, the term could refer to an interchange box, a link, a multiplexer, or a demultiplexer.

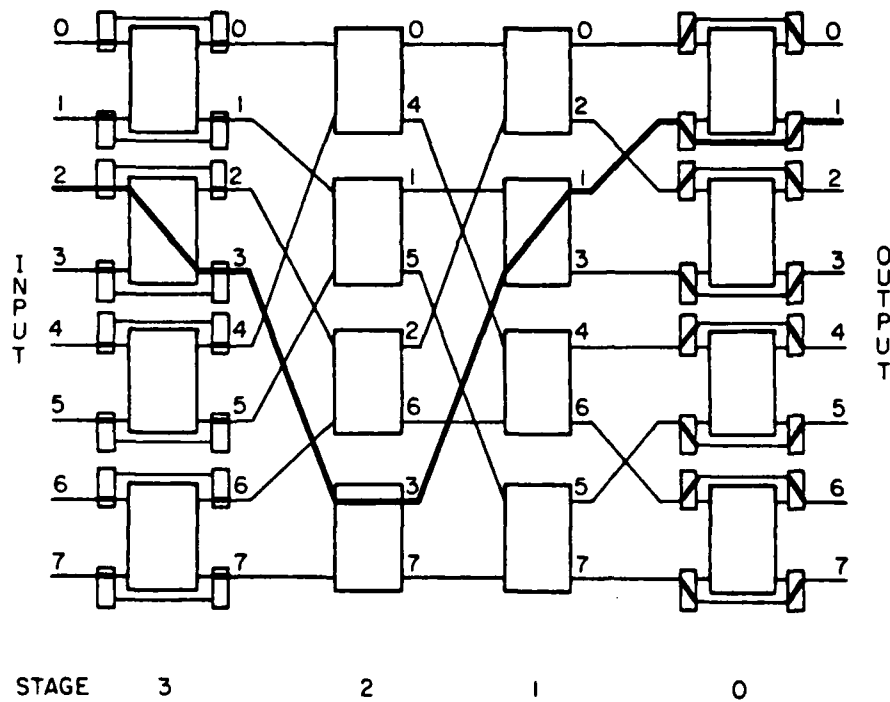


Figure 3.6 The path from input 2 to output 1 in the ESC for $N=8$, when stage n is enabled and stage 0 is disabled. Stage n and 0 multiplexer and demultiplexer settings are shown explicitly [Sie85].

3.5 Conclusions

This chapter has presented the Extra Stage Cube interconnection network. The Generalized Cube network was defined and the derivation of the ESC from it was shown. The ESC is only slightly more physically complex than its parent network. Properties of the Generalized Cube, which are also properties of the ESC, were listed. Its extra stage of switching elements and its bypass circuitry allow the ESC to tolerate faults, a property not shared with the Generalized Cube network. Operation of ESC bypass circuitry was described.

CHAPTER 4

PROPERTIES OF THE EXTRA STAGE CUBE INTERCONNECTION NETWORK

4.1 Introduction

The discussion in Chapter 2 of parallel systems has demonstrated that an important component of a parallel computer is a mechanism for information transfer between the various system components. Because of system complexity, assuring high reliability is a significant task. Thus, a crucial aspect of an interconnection network used to meet system communication needs is fault tolerance.

The formal description of the fault tolerance of an interconnection network requires the definition of both a fault model and fault tolerance criterion. With these in hand the fault tolerance of the ESC will be studied. After considering the effects of a single fault, the response of the ESC to multiple faults is considered as well, because of the desirability of graceful degradation capability. Other important aspects of any interconnection network include: method of control, routing capability, partitioning capability, and permuting capability. Each of these issues is explored in this chapter. All will be related to the fault tolerance of the ESC, demonstrating the feasibility of achieving fault tolerant communication for a parallel computer with this network.

4.2 Fault-Tolerance Model

Networks that can continue, in at least some cases, to provide interconnection service even when they contain faulty components are said to be *fault-tolerant*. A network is termed *single-fault tolerant* if it can function in spite of a single fault. If up to i faults can be tolerated then the network is *i -fault tolerant*. A network is *robust* if it can tolerate some instances of i faults, but is not i -fault tolerant. A fault is *hard* if it is not of a transient nature; all faults will be assumed hard unless otherwise stated.

It is only meaningful to speak of a network as i -fault tolerant with regard to a particular *fault-tolerance model*. A fault-tolerance model consists of two components. The first is the *fault model*, which characterizes all faults that are assumed to occur in the network. The fault model for a given network may or may not correspond closely to actual or predicted experience with hardware. In particular, fault models are often chosen to have characteristics suited for performing an analysis, even if those characteristics may depart widely from reality. The second component is the *fault-tolerance criterion*, the condition that must be met for the network to be said to have tolerated a given fault or faults. This criterion varies due to differences in the definition of what constitutes functionality for a given network (basically, what amount of degradation from the fault-free condition is allowed).

The fault model chosen for the ESC is the following.

1. Any interchange box or link can fail.

2. Faulty boxes and links are considered unusable until such time as the fault is remedied.
3. Faults occur independently.
4. All interchange boxes have identical reliability, which may differ from the identical reliability characterizing all links.
5. Input stage multiplexers and output stage demultiplexers have no effect on the reliability of their associated links.

Item 1 of the fault model implies that network ports, input demultiplexers, and output multiplexers are always fault-free. Devices cannot access the network if these components fail. Chapter 7 shows that physical implementation of the ESC need not involve input demultiplexers nor output multiplexers.

The ESC fault-tolerance criterion is retention of full access [CiS82]. *Full access* capability is the ability to connect any given input to any output, a property of the fault-free network. This implies that the ESC with a single fault retains its *fault-free interconnection capability*.

Once a fault has been detected and located in the ESC, the failing portion of the network is considered unusable until such time as the fault is remedied. Specifically, if an interchange box is faulty, data will not be routed through it, nor will data be passed over a faulty link. The extra stage of the ESC does increase the likelihood of a fault, compared to the Generalized Cube, due to the additional hardware. It should also be noted that a failure in a stage n multiplexer or stage 0 demultiplexer has the effect of a link fault, which the ESC can tolerate, as shown in Section 4.3.

Techniques such as test patterns [FeK82, FeW81, Lim82, LeS83], dynamic parity checking [SiM81b, LLY82], or write/read-back/verify [ThN83] for fault

detection and location have been described for use in the Generalized Cube topology. Test patterns are used to determine network integrity globally by checking the data arriving at the network outputs as a result of N strings (one per input port) of test inputs. With dynamic parity checking, each interchange box monitors the status of boxes and links connected to its inputs by examining incoming data for correct parity. It is assumed that the ESC can be tested to determine the existence and location of faults. This chapter is not concerned with the procedures to accomplish this, but rather with how to recover once a fault is located. Recovery from such a fault is something of which the Generalized Cube and its topologically equivalent networks are incapable. The Generalized Cube is neither single-fault tolerant nor robust with respect to the ESC fault model and fault-tolerance criterion.

4.3 Single-Fault Tolerance

The ESC achieves its fault-tolerant capabilities by having redundant paths and broadcast paths. This section formally establishes the existence of these redundant paths. Then procedures for finding a fault-free path or broadcast path in a faulted, but functional, ESC are set forth.

4.3.1 One-to-One Connections

The first step towards establishing the fault tolerance of the ESC is to show that it can provide more than one path between any source and destination.

Theorem 4.1: In the ESC with both stages n and 0 enabled there exist exactly two paths between any source and any destination.

Proof: There is exactly one path from a source S to a destination D in the Generalized Cube [Law75]. Stage n of the ESC allows access to two distinct stage $n-1$ inputs, S and $\text{cube}_0(S)$. Stages $n-1$ to 0 of the ESC form a Generalized Cube topology, so the two stage $n-1$ inputs each have a single path to the destination and these paths are distinct (since they differ at stage $n-1$ at least).

□

Figure 4.1 shows the two paths available between input 1 and output 4 in an ESC with $N=8$ when both stages n and 0 are enabled. The settings of the stage n and 0 multiplexers and demultiplexers are shown to explicitly depict stages n and 0 enabled. One path from 1 to 4 uses the straight setting in stage n ; the other uses the exchange setting.

The existence of at least two paths between any source/destination pair is a necessary condition for fault tolerance. Redundant paths allow continued communication between source and destination if at least one path remains functional after a fault. It can be shown that for the ESC two paths are sufficient to provide tolerance to single faults for one-to-one connections.

Lemma 4.1: The two paths between a given source and destination in the ESC with stages n and 0 enabled have no links in common.

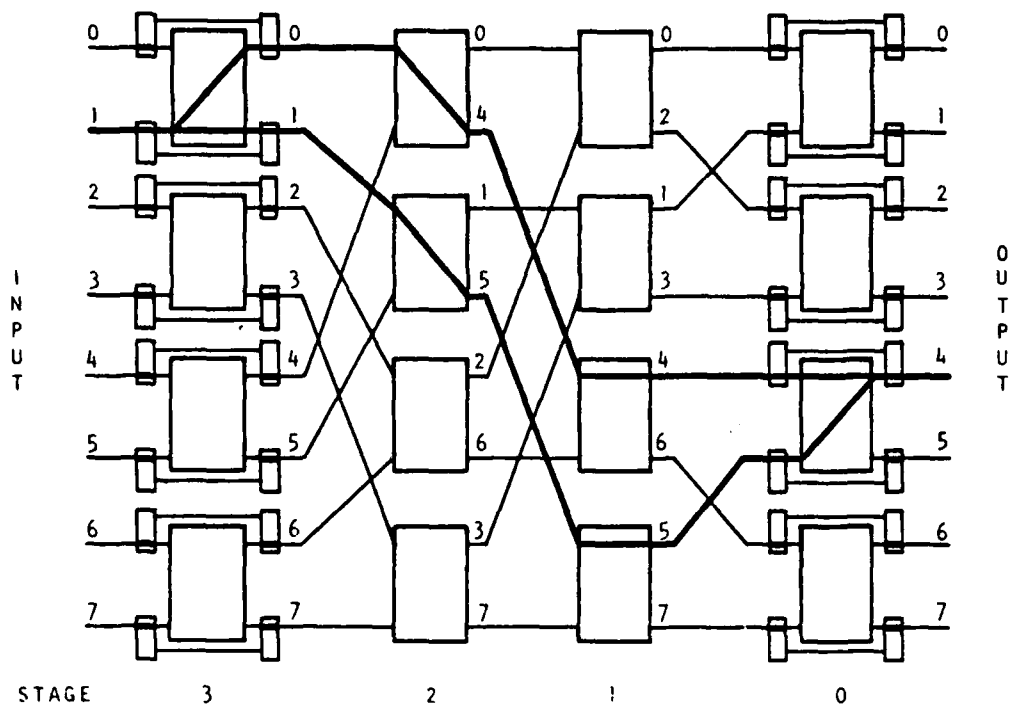


Figure 4.1 The two paths in the ESC network for $N=8$ between input 1 and output 4 when both stages n and 0 are enabled. Stage n and 0 multiplexer and demultiplexer settings are shown explicitly.

Proof: A source S can connect to the stage $n-1$ inputs S or $\text{cube}_0(S)$. These two inputs differ in the 0^{th} , or low-order, bit position. Other than stage n , only stage 0 can cause a source to be mapped to a destination which differs from the source in the low-order bit position. Therefore, the path from S through stage $n-1$ input S to the destination D contains only links with labels which agree with S in the low-order bit position. Similarly, the path through stage $n-1$ input $\text{cube}_0(S)$ contains only links with labels agreeing with $\text{cube}_0(S)$ in the low-order bit position. Thus, no link is part of both paths.

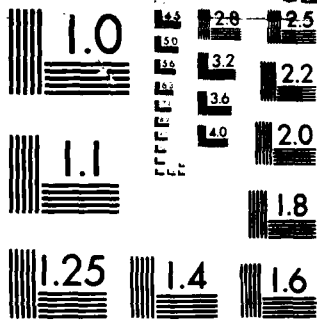
□

Lemma 4.2: The two paths between a given source and destination in the ESC with stages n and 0 enabled have no interchange boxes from stage $n-1$ through 1 in common.

Proof: Since the two paths have the same source and destination, they will pass through the same stage n and 0 interchange boxes. No box in stages $n-1$ through 1 has input link labels that differ in the low-order bit position. One path from S to D contains only links with labels agreeing with S in the low-order bit position. The other path has only links with labels that are the complement of S in the low-order bit position. Therefore, no box in stages $n-1$ through 1 belongs to both paths.

□

Theorem 4.2: In the ESC with a single fault there exists at least one fault-free path between any source and destination.



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

Proof: Assume first that a link is faulty. If both stages n and 0 are enabled, Lemma 4.1 implies that at most one of the paths between a source and destination can be faulty. Hence, a fault-free path exists.

Now assume that an interchange box is faulty. There are two cases to consider. If the faulty box is in stage n or 0 , the affected stage can be disabled. The remaining n stages are sufficient to provide one path between any source and destination (i.e., all n cube functions are still available). If the faulty box is not in stage n or 0 , Lemma 4.2 implies that if both stages n and 0 are enabled then at most one of the paths is faulty. So, again, a fault-free path exists.

Two paths exist when the fault is in neither of the two paths between source and destination.

□

Figure 4.2 shows one example of how the ESC tolerates a single fault. A stage 2 link is indicated as faulty by the dashed line. The two possible paths from stage n when it is enabled are shown, since it will be enabled due to the existence of a fault in a network component other than a stage n box. Despite the fact that stage 0 is bypassed as a result of the fault, one of these paths remains fault-free.

4.3.2 Broadcast Connections

The two paths between any source and destination of the ESC provide fault tolerance for performing broadcasts as well.

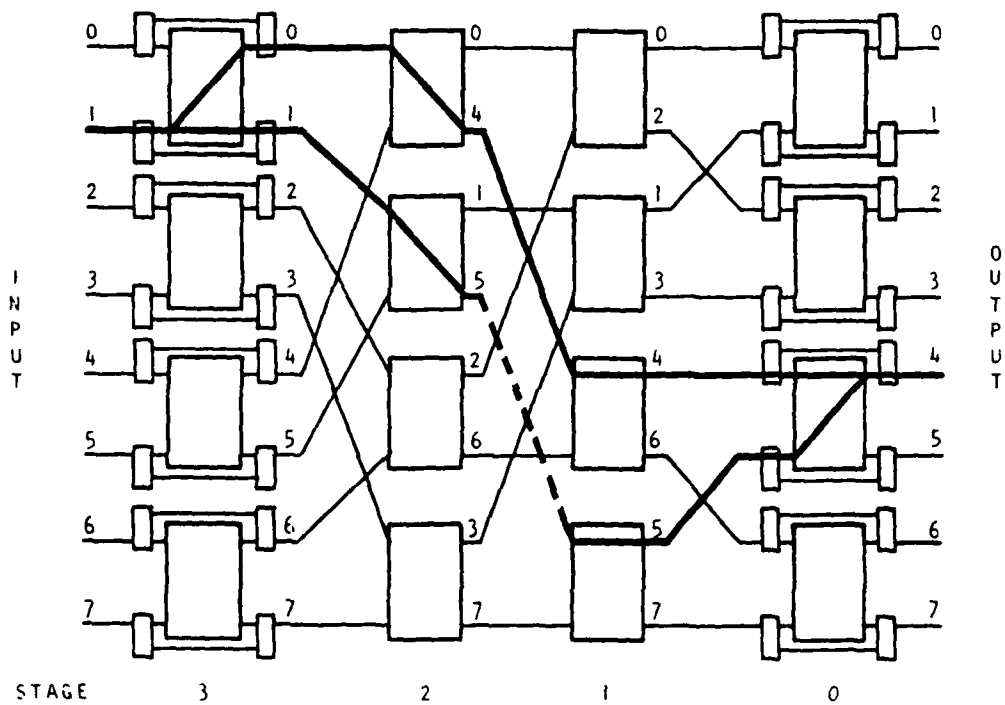


Figure 4.2 ESC network with $N=8$ and a faulty stage 2 link (indicated by the broken line) showing the two paths from input 1 to output 4, one of which is fault-free.

Theorem 4.3: In the ESC with both stages n and 0 enabled there exist exactly two broadcast paths for any broadcast.

Proof: A one-to-many broadcast path is just a collection of one-to-one connections with the same source. There is exactly one broadcast path from a source to its destinations in the Generalized Cube [SiS78, SiM81b]. Stage n of the ESC allows a source S access to two distinct stage $n-1$ inputs, S and $\text{cube}_0(S)$. Any set of destinations to which S can broadcast, $\text{cube}_0(S)$ can also broadcast, because any broadcast can be performed.

□

Figure 4.3 illustrates the two broadcast paths available to connect input 0 to outputs $2, 3, 6,$ and 7 in an ESC with $N=8$ when both stage n and 0 are enabled. The bold line indicates one complete broadcast path, the dashed line denotes the other.

Lemma 4.3: The two broadcast paths between a given source and its destinations in the ESC with stages n and 0 enabled have no links in common.

Proof: All links in the broadcast path from the stage $n-1$ input S have labels which agree with S in the low-order bit position. All links in the broadcast path from the stage $n-1$ input $\text{cube}_0(S)$ are the complement of S in the low-order bit position. Thus, no link is part of both broadcast paths.

□

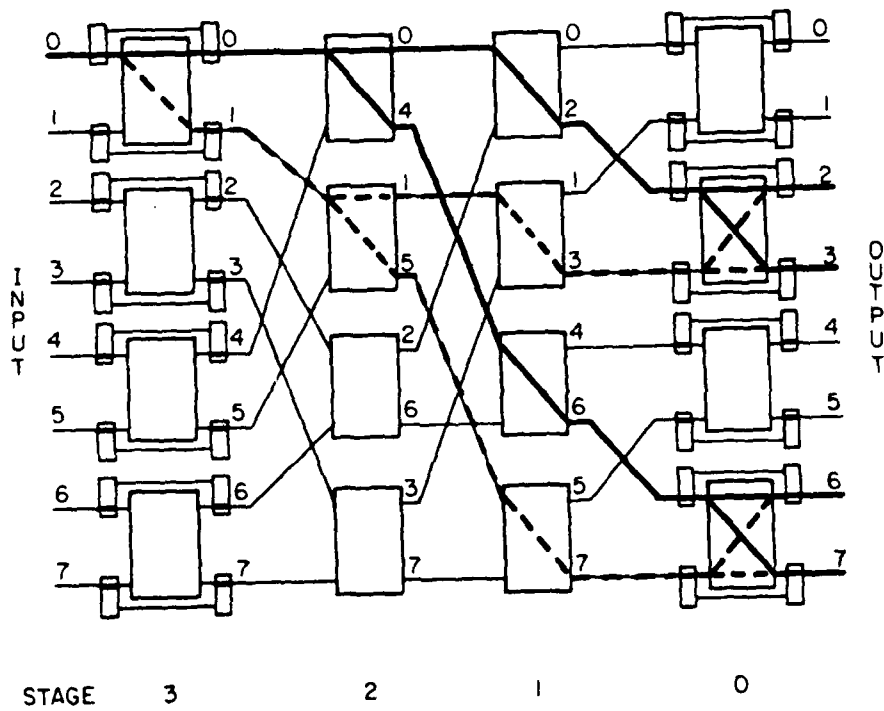


Figure 4.3 The two paths from input 0 to outputs 2, 3, 6, and 7 in the ESC network for $N=8$. The bold lines indicate one complete broadcast path, the dashed lines denote the other [Sie85].

Lemma 4.4: The two broadcast paths between a given source and its destinations in the ESC with stages n and 0 enabled have no interchange boxes from stage $n-1$ through 1 in common.

Proof: Since the two broadcast paths have the same source and destinations, they will pass through the same stage n and 0 interchange boxes. No box in stages $n-1$ through 1 has input link labels which differ in the low-order bit position. From the proof of Lemma 4.3, the link labels of the two broadcast paths differ in the low-order bit position. Therefore, no box in stages $n-1$ through 1 belongs to both broadcast paths.

□

Lemma 4.5: With stage 0 disabled and stage n enabled, the ESC can form any broadcast path which can be formed by the Generalized Cube.

Proof: Stages n through 1 of the ESC provide a complete set of n cube interconnection functions in the order $\text{cube}_0, \text{cube}_{n-1}, \dots, \text{cube}_1$. A path exists between any source and destination with stage 0 disabled because all n cube functions are available. This is regardless of the order of the interconnection functions. So, a set of paths connecting an arbitrary source to any set of destinations exists. Therefore, any broadcast path can be formed.

□

Theorem 4.4: In the ESC with a single fault there exists at least one fault-free broadcast path for any broadcast performable by the Generalized Cube.

Proof: Assume the fault is in stage 0, i.e., disable stage 0, enable stage n. Lemma 4.5 implies a fault-free broadcast path exists. Assume the fault is in a link or a box in stages n-1 to 1. From Lemmas 4.3 and 4.4, the two broadcast paths will have none of these network elements in common. Therefore, at least one broadcast path will be fault-free, possibly both. Finally, assume the fault is in stage n. Stage n will be disabled and the broadcast capability of the ESC will be the same as that of the Generalized Cube.

□

4.3.3 Finding Fault-Free Paths

The ESC path that routes S to D and corresponds to the Generalized Cube path from S to D is called the *primary path*. This path must either bypass stage n or use the straight setting in stage n. The other path available to connect S to D is the *secondary path*. It must use the exchange setting in stage n. The concept of primary path can be extended for broadcasting. The broadcast path, or set of paths, in the ESC analogous to that available in the Generalized Cube is called the *primary broadcast path*. This is because each path from the source to one of the destinations is a primary path. If every primary path is replaced by its secondary path the result is the *secondary broadcast path*.

Given S and D, the network links and boxes used by a path can be found. As discussed in [Law75], for the source/destination pair S with binary representation $s_{n-1} \dots s_1 s_0$ and D with binary representation $d_{n-1} \dots d_1 d_0$ the path followed in the Generalized Cube topology uses the stage i output labeled $d_{n-1} \dots d_{i+1} d_i s_{i-1} \dots s_1 s_0$. The following theorem extends this for the ESC. Note that given $x_{n-1} \dots x_1 x_0$ and $y_{n-1} \dots y_1 y_0$, the notational convention that

$x_{n-1} \dots x_{i+1} x_i y_{i-1} \dots y_1 y_0 = y_{n-1} \dots y_1 y_0$, if $i = n$, is used. Also,
 $x_{n-1} \dots x_{i+1} x_i y_{i-1} \dots y_1 y_0 = x_{n-1} \dots x_1 x_0$, if $i = 0$.

Theorem 4.5: For the source/destination pair $S = s_{n-1} \dots s_1 s_0$ and $D = d_{n-1} \dots d_1 d_0$, the primary path uses the stage i output labeled $d_{n-1} \dots d_{i+1} d_i s_{i-1} \dots s_1 s_0$ and the secondary path uses $d_{n-1} \dots d_{i+1} d_i s_{i-1} \dots s_1 \bar{s}_0$, for $0 \leq i \leq n$.

Proof: Stage i , $i \neq 0$, is the only stage in the ESC that can map the i^{th} bit of a source address (i.e., determine the i^{th} bit of the destination). Thus, if S is to reach D both ESC paths must use a stage i output with a label that matches D in the i^{th} bit position. This matching occurs at each stage, so the high-order $n-i$ bits of the output label will be $d_{n-1} \dots d_{i+1} d_i$. At the output of stage i , bit positions $i-1$ to 1 have yet to be affected so they match source address bits $i-1$ to 1 . The low-order bit position is unchanged by stage n for the primary path. The secondary path includes the cube_0 connection (exchange) in stage n , therefore the low-order bit position is complemented.

□

When a fault has been detected and located, each source will receive a *fault label*, or labels, uniquely specifying the fault location. This is accomplished by giving a *stage number* and a *stage output number*. For example, if the link between stages i and $i-1$ from the stage i output j fails, each source receives the fault label (i,j) . If a box in stage i with outputs j and k fails, the pair of fault labels (i,j) and (i,k) is sent to each source. Since j and k differ only in the i^{th} bit position a single fault label $(i, h_{n-1} \dots h_{i+1} x h_{i-1} \dots h_1 h_0)$

can be sent, where $h_m = j_m = k_m$ for $0 \leq m \leq n-1$ and $m \neq i$. The "x" represents both a 1 and a 0. For a fault in stage 0 no fault label will be given, only notice that a stage 0 fault exists. This is because stage 0 will be disabled in the event of such a fault, so no path could include a stage 0 fault. Note that the only possible faults in stage 0 are box faults. Stage n box faults require system maintenance, but no fault labels need be issued, as the stage will be disabled. A label is issued in the event of a stage n link fault. Note also that the number of faults may not equal the number of fault labels.

If a network fault lies outside stage n or 0 boxes, a source can check the primary path to its intended destination for the faulty network component. For any faulty link or a faulty box in stage i, $1 \leq i < n$, the source forms $d_{n-1} \dots d_{i+1} d_i s_{i-1} \dots s_1 s_0$ and compares this with the stage output numbers of the fault label(s). (Note that if there is a faulty stage n link $s_{n-1} \dots s_1 s_0$ is used.) If there is a match then the primary path is faulty; if not, it is fault-free. If the primary path is fault-free it can be used. If faulty, the secondary path will be fault-free and, hence, usable.

Since a broadcast path to many destinations is the union of many paths from the source to the destinations, exhaustive checking to see if one of these paths contains a fault may involve more computational effort than is desirable. To decide if the secondary broadcast path should be used, a simpler criterion than checking each path for the fault exists. For any faulty link or a faulty box in stage i, $1 \leq i < n$, the test is to compare the low-order i bits of the source address and the stage output number(s) of the faulty link or box. The primary broadcast paths must use links and stage n-1 to 1 boxes with stage output numbers that agree with the source address in the low-order bit positions. Thus, if the low-order i bits of the fault label stage output

number(s) and the source address agree, the fault may lie in the primary broadcast path. (The fault is in the primary broadcast path if the high-order $n-i$ bits of the stage output number match the corresponding bits of any destination of the broadcast.) Using the secondary broadcast path avoids the possibility of encountering the fault. This method is computationally simpler than exhaustive path fault checking, but can result in unneeded use of the secondary broadcast path.

If there is no strong preference to using the primary versus secondary path (or broadcast path), the test to check for any faulty link or stage i box, $1 \leq i < n$, can be reduced to just comparing on a single bit position. If the low-order source address and fault label stage output number bits agree, then the primary path (or primary broadcast path) may be faulty, so the secondary path can be used. This simplified procedure will result in unnecessary use of secondary paths (one-to-one), and more unnecessary use of secondary broadcast paths than checking the low-order i bits.

4.4 Multiple-Fault Tolerance

Theorems 4.2 and 4.4 establish the capability of the ESC to tolerate a single fault given its fault-tolerance model. That is, any one-to-one or broadcast connection possible in the fault-free Generalized Cube network remains possible in the ESC despite a single fault. For some instances of multiple faults the ESC also retains fault-free interconnection capability. The necessary and sufficient condition for this is that the primary and secondary paths are not both faulty.

As faults are detected and located a system control unit can determine whether network interconnection capability is degraded.

Theorem 4.6: Let $A = (i, a_{n-1} \dots a_1 a_0)$ and $B = (j, b_{n-1} \dots b_1 b_0)$, where $1 \leq j \leq i \leq n$, be two fault labels. If $a_{n-1} \dots a_{i+1} a_i \neq b_{n-1} \dots b_{i+1} b_i$, or if $a_{j-1} \dots a_1 \bar{a}_0 \neq b_{j-1} \dots b_1 b_0$, then there will be at least one fault-free path between any source and destination.

Proof: A fault-free path will exist for a source/destination pair S/D if taken together the fault labels A and B do not indicate blockage of both the primary and secondary paths. As shown in Theorem 4.5, the primary path uses stage i output $d_{n-1} \dots d_{i+1} d_i s_{i-1} \dots s_1 s_0$ and the secondary path uses $d_{n-1} \dots d_{i+1} d_i s_{i-1} \dots s_1 \bar{s}_0$. The stage j outputs used are $d_{n-1} \dots d_{j+1} d_j s_{j-1} \dots s_1 s_0$ and $d_{n-1} \dots d_{j+1} d_j s_{j-1} \dots s_1 \bar{s}_0$. Without loss of generality it is assumed that $j \leq i$. Thus, at stages i and j the primary and secondary paths both use outputs with the same bits in positions $n-1$ through i and $j-1$ through 1, and complementary values in position 0. If $a_{n-1} \dots a_{i+1} a_i \neq b_{n-1} \dots b_{i+1} b_i$ then at least one of the faults is in neither the primary nor the secondary path, so at least one of the paths is fault-free. Similarly, if $a_{j-1} \dots a_1 \bar{a}_0 \neq b_{j-1} \dots b_1 b_0$ at least one fault is in neither path, so at least one path is fault-free. For $i = n$ the inequality $a_{n-1} \dots a_{i+1} a_i \neq b_{n-1} \dots b_{i+1} b_i$ does not exist, and only the constraint $a_{j-1} \dots a_1 \bar{a}_0 \neq b_{j-1} \dots b_1 b_0$ applies.

□

When multiple faults are detected and located in the ESC, a system control unit must determine the appropriate action. Theorem 4.6 is applied if the multiple faults occur in links or in boxes in stages $n-1$ to 1. The fault label(s) of any new fault(s) is compared with all existing fault label(s). If each pair of fault labels meets the test of Theorem 4.6, then the network retains its

fault-free interconnection capability. (Note that the two fault labels associated with a faulty box do satisfy the requirement of Theorem 4.6 since for stages $n-1$ through 1 the low-order bits of the stage output numbers of such labels agree, satisfying the Theorem 4.6 criterion $a_{j-1} \dots a_1 \bar{a}_0 \neq b_{j-1} \dots b_1 b_0$.)

With multiple stage 0 or multiple stage n faults only, the affected stage is simply disabled, as for a single fault in either stage; fault-free interconnection capability still exists. If faults exist in boxes of both stages n and 0 , the stages are both disabled, and the network cannot perform cube_0 . Thus, fault-free interconnection capability is lost. If there are faults in stages $n-1$ through 1 and either stage n or 0 boxes, but not both, complete fault-free interconnection capability no longer exists. This is because only one of stages n and 0 will be available to be enabled, hence, there is only one path between and source and destination. So, any faulty links or faulty boxes in stage i , $1 \leq i < n$, will block the only path between certain source/destination pairs. Finally, if a pair of fault labels fails the test of Theorem 4.6, fault-free interconnection capability is lost.

Figure 4.4 shows an ESC network with $N = 8$ and two faults which do not cause a loss of fault-free interconnection capability. The dashed lines indicate the two faults, which have fault labels $(2,2)$ and $(1,4)$. The bold lines indicate all possible paths through the network that contain the fault $(2,2)$. The paths are shown using only the connections in the interchange boxes to simplify the figure. The bold lines should be taken to denote the two alternative connections through each box that could be part of a path containing fault $(2,2)$. The dotted lines are analogous to the bold lines, but for fault $(1,4)$. Note that no source has both possible stage n outputs leading to fault $(2,2)$ or $(1,4)$, and no destination has both possible stage 0 inputs coming from the

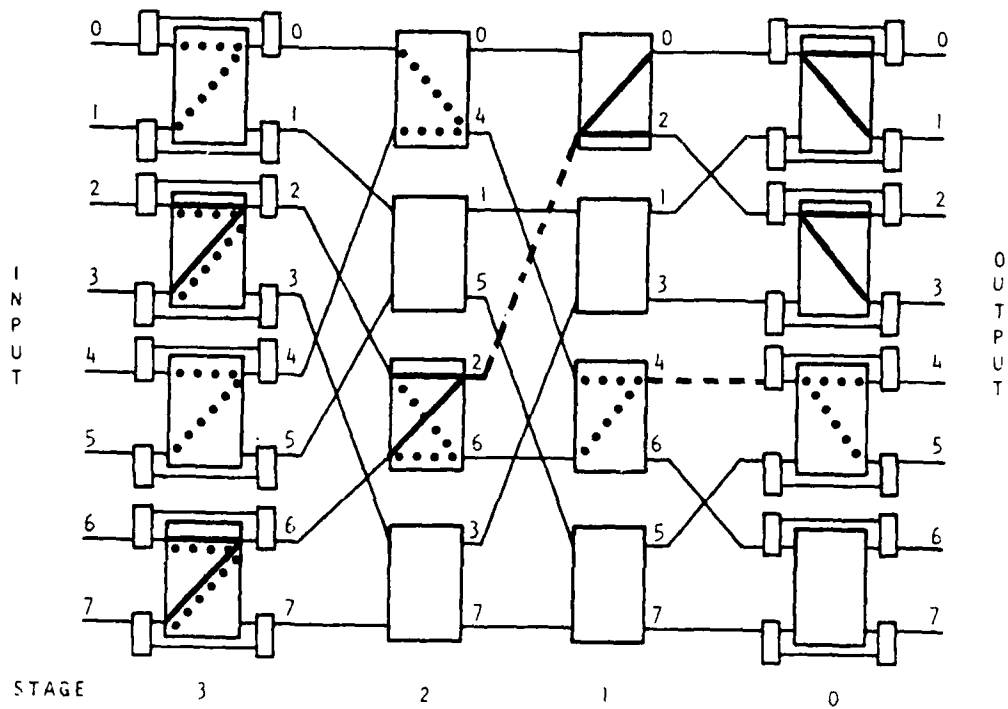


Figure 4.4 ESC network with $N=8$ and faults (2,2) and (1,4) (indicated by broken lines), which do not cause a loss of fault-free interconnection capability.

faults. Therefore, no matter what destination is paired with any source there will exist a fault-free connecting path.

If fault-free interconnection capability exists then full operation can continue. To continue, any additional fault labels are sent to each source. However, a source must now check a primary path against a longer list of fault labels to determine if that path is fault-free. Therefore, system performance may be degraded somewhat.

For an SIMD system where interconnection network routing requirements are limited to a relatively small number of known mappings, multiple faults that preclude fault-free interconnection capability might not impact system function. This would occur if all needed permutations could be performed (although each would require two passes (see Section 4.7)). Similar faults in MSIMD or MIMD systems may leave some processes unaffected. For these situations, and if fail-soft capability is important, it is useful to determine which source/destination pairs are unable to communicate. The system might then attempt to reschedule processes such that their needed communication paths will be available, or assess the impact the faults will have on its performance and report to the user.

The exact conditions under which no fault-free path exists, for some one-to-one connection, can be determined and the affected source/destination pairs characterized.

Corollary 4.1: Let $A = (i, a_{n-1} \dots a_1 a_0)$ and $B = (j, b_{n-1} \dots b_1 b_0)$, where $1 \leq j \leq i \leq n$, be two fault labels. If $a_{n-1} \dots a_{i+1} a_i = b_{n-1} \dots b_{i+1} b_i$ and $a_{j-1} \dots a_1 \bar{a}_0 = b_{j-1} \dots b_1 b_0$ then there exist source/destination pairs for which no fault-free path exists. These pairs are such that $s_{i-1} \dots s_2 s_1 = a_{i-1} \dots a_2 a_1$,

$d_{n-1} \dots d_{j+1} d_j = b_{n-1} \dots b_{j+1} b_j$, and $s_{n-1} \dots s_{i+1} s_i$, s_0 , and $d_{j-1} \dots d_1 d_0$ are arbitrary.

Proof: From Theorem 4.5, the two paths between a source and destination use stage k outputs which differ only in the low-order bit, $1 \leq k \leq n$. Assume a path contains the fault denoted by A. Then the stage i output used is such that $d_{n-1} \dots d_{i+1} d_i s_{i-1} \dots s_2 s_1 x = a_{n-1} \dots a_1 a_0$ and the stage j output used is $d_{n-1} \dots d_{j+1} d_j s_{j-1} \dots s_2 s_1 x$ where x equals s_0 or \bar{s}_0 depending on whether the path is primary or secondary. Now $a_{n-1} \dots a_{i+1} a_i = b_{n-1} \dots b_{i+1} b_i$ and $a_{j-1} \dots a_1 \bar{a}_0 = b_{j-1} \dots b_1 b_0$. Thus, the alternate path uses stage j output $d_{n-1} \dots d_{j+1} d_j s_{j-1} \dots s_2 s_1 \bar{x} = a_{n-1} \dots a_{i+1} a_i d_{i-1} \dots d_{j+1} d_j$ $a_{j-1} \dots a_1 \bar{a}_0 = b_{n-1} \dots b_{i+1} b_i d_{i-1} \dots d_{j+1} d_j b_{j-1} \dots b_1 b_0$. If $d_{i-1} \dots d_{j+1} d_j = b_{i-1} \dots b_{j+1} b_j$, then the alternate path contains the fault denoted by B. Therefore, there exist source/destination pairs for which no fault-free path exists.

The relationships $d_{n-1} \dots d_{i+1} d_i s_{i-1} \dots s_2 s_1 x = a_{n-1} \dots a_1 a_0$ and $d_{n-1} \dots d_{j+1} d_j s_{j-1} \dots s_2 s_1 \bar{x} = b_{n-1} \dots b_1 b_0$ yield the constraints on the source and destination addresses of $s_{i-1} \dots s_2 s_1 = a_{i-1} \dots a_2 a_1$ and $d_{n-1} \dots d_{j+1} d_j = b_{n-1} \dots b_{j+1} b_j$. The values of $s_{n-1} \dots s_{i+1} s_i$, s_0 , and $d_{j-1} \dots d_1 d_0$ are unconstrained. □

Figure 4.5 shows an ESC with $N=8$ for a case of two faults with fault labels which fail the Theorem 4.6 criterion. The affected sources and destinations are indicated by the circles. The affected sources (0, 1, 4, and 5) have paths that can lead to one of the faults on each of their stage n outputs. The affected destinations (4, 5, 6, and 7) have paths that can come from one of the faults on each of their stage 0 inputs. An affected source cannot communicate with affected destinations, but can with unaffected ones.

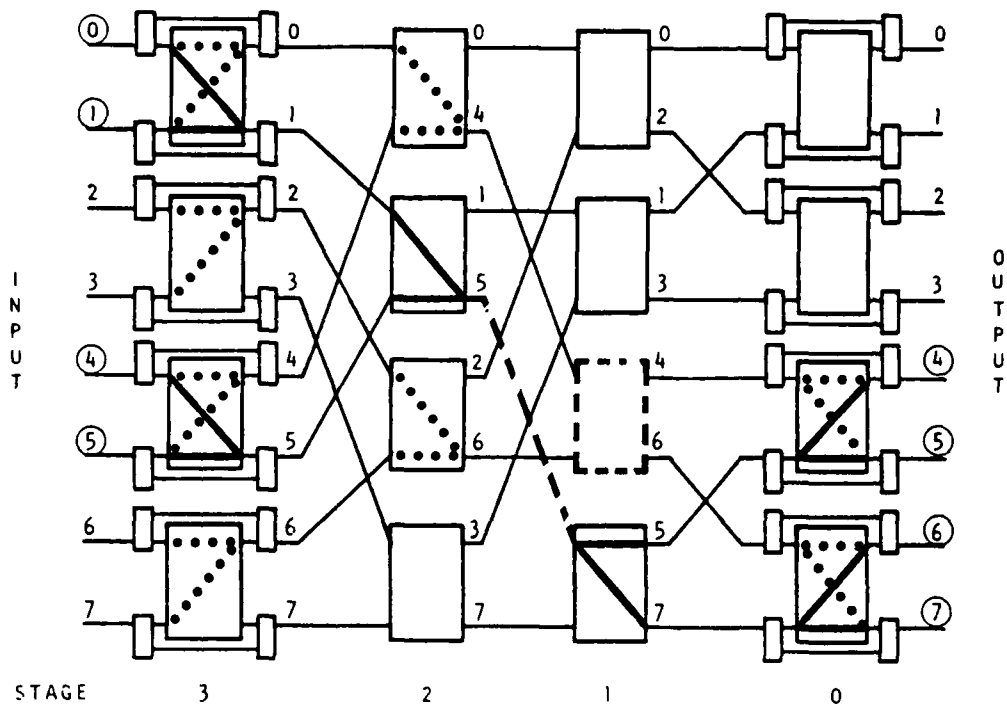


Figure 4.5 ESC network with $N=8$ and faults (2,5), (1,4), and (1,6) (indicated by broken lines), which cause a loss of fault-free interconnection capability. The affected network inputs and outputs are circled.

Similarly, affected destinations can receive information from unaffected sources, and not affected ones. Unaffected sources and destinations can communicate as if the network were fault-free.

For broadcast paths, continued operation under multiple faults is somewhat more complicated since faults can exist in both a primary and secondary broadcast path without compromising fault-free (one-to-one) interconnection capability. The tests to determine if a primary path contains a fault, which were described in Section 4.3.3, can be applied in this case. To check for a possible fault in a secondary path, \bar{s}_0 is used in place of s_0 . If both paths contain faults, a combination of primary and secondary paths can be used to perform the broadcast. However, this procedure may be too time consuming to be practical.

The exact conditions under which no fault-free broadcast path exists, for some broadcast, can be determined and the affected broadcasts characterized.

Corollary 4.2: Let $A = (i, a_{n-1} \dots a_1 a_0)$ and $B = (j, b_{n-1} \dots b_1 b_0)$, where $1 \leq j \leq i \leq n$, be any two fault labels. If $a_{j-1} \dots a_1 \bar{a}_0 = b_{j-1} \dots b_1 b_0$, then there exist broadcasts for which no fault-free broadcast path exists. These broadcasts are such that $s_{i-1} \dots s_2 s_1 = a_{i-1} \dots a_2 a_1$, $d_{n-1}^k \dots d_{i+1}^k d_i^k = a_{n-1} \dots a_{i+1} a_i$, and $d_{n-1}^l \dots d_{j+1}^l d_j^l = b_{n-1} \dots b_{j+1} b_j$, where $S = s_{n-1} \dots s_1 s_0$ is the source and $D^k = d_{n-1}^k \dots d_j^k d_0^k$ and $D^l = d_{n-1}^l \dots d_j^l d_0^l$ are two of the destinations (and are not necessarily distinct).

Proof: In general, a broadcast path uses stage i outputs of the form $d_{n-1} \dots d_{i+1} d_i s_{i-1} \dots s_2 s_1 x$, where x equals s_0 or \bar{s}_0 depending on whether the broadcast path is primary or secondary, and $D = d_{n-1} \dots d_j d_0$ represents one of

the broadcast destinations. As a consequence of Theorem 4.5, the two broadcast paths from a source to a set of destinations use stage m outputs which differ only in the low-order bit, where $1 \leq m \leq n$. Thus, the alternate broadcast path uses stage j outputs of the form $d_{n-1} \dots d_{j+1} d_j s_{j-1} \dots s_2 s_1 \bar{x}$. To construct those broadcasts with faulty primary and secondary paths due to faults A and B, consider the following. Let the source $S = s_{n-1} \dots s_1 s_0$ be such that $s_{i-1} \dots s_2 s_1 = a_{i-1} \dots a_2 a_1$, and, without loss of generality let $x = a_0$. Let $D^k = d_{n-1}^k \dots d_j^k d_0^k$, one of the destinations, be such that $d_{n-1}^k \dots d_{i+1}^k d_i^k = a_{n-1} \dots a_{i+1} a_i$. Let $D^l = d_{n-1}^l \dots d_j^l d_0^l$, another destination (not necessarily distinct from D^k), be such that $d_{n-1}^l \dots d_{j+1}^l d_j^l = b_{n-1} \dots b_{j+1} b_j$. Given $a_{j-1} \dots a_1 \bar{a}_0 = b_{j-1} \dots b_1 b_0$, then the equalities $d_{n-1}^k \dots d_{i+1}^k d_i^k s_{i-1} \dots s_2 s_1 x = a_{n-1} \dots a_1 a_0$ and $d_{n-1}^l \dots d_{j+1}^l d_j^l s_{j-1} \dots s_2 s_1 \bar{x} = b_{n-1} \dots b_1 b_0$ are true. Any broadcast for which the equalities hold does not have a fault-free primary or secondary broadcast path.

□

Figures 4.6, 4.7, and 4.8 show how a broadcast may have to be performed given multiple faults, even though fault-free interconnection capability is retained. Figure 4.6 depicts an ESC for $N=8$ with faults (2,2) and (1,5), which do not cause loss of fault-free interconnection capability, and the primary broadcast path for the broadcast from input 2 to outputs 1, 3, 5, and 7. Stages n and 0 are enabled due to the faults. The primary broadcast path contains the fault (2,2). Figure 4.7 shows the secondary broadcast path which contains fault (1,5). A broadcast path combining fault-free portions of the primary and secondary broadcast paths is indicated in Figure 4.8.

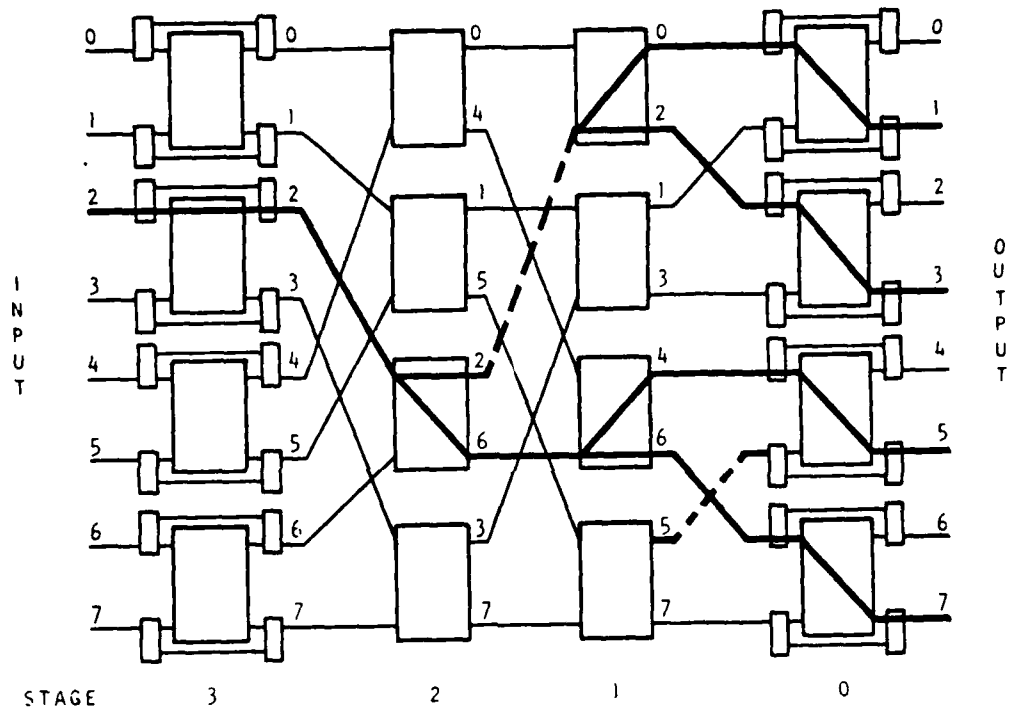


Figure 4.6 ESC network with $N=8$ and faults (2,2) and (1,5) showing the primary broadcast path from input 2 to outputs 1, 3, 5, and 7. This path contains fault (2,2).

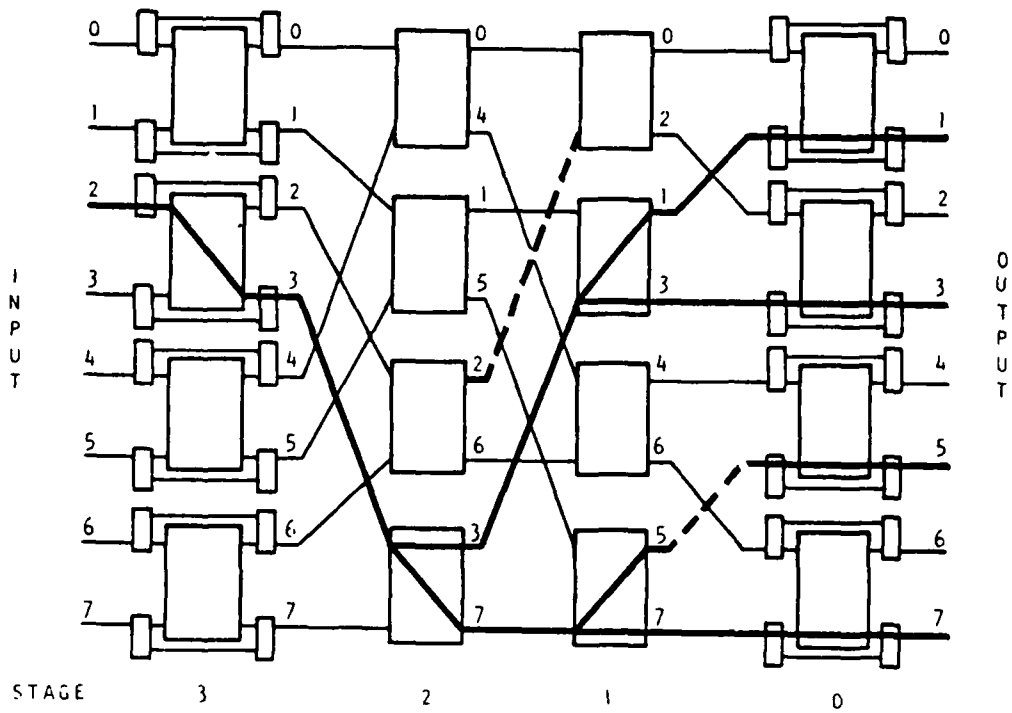


Figure 4.7 ESC network with $N=8$ and faults (2,2) and (1,5) showing the secondary broadcast path from input 2 to outputs 1, 3, 5, and 7. This path contains fault (1,5).

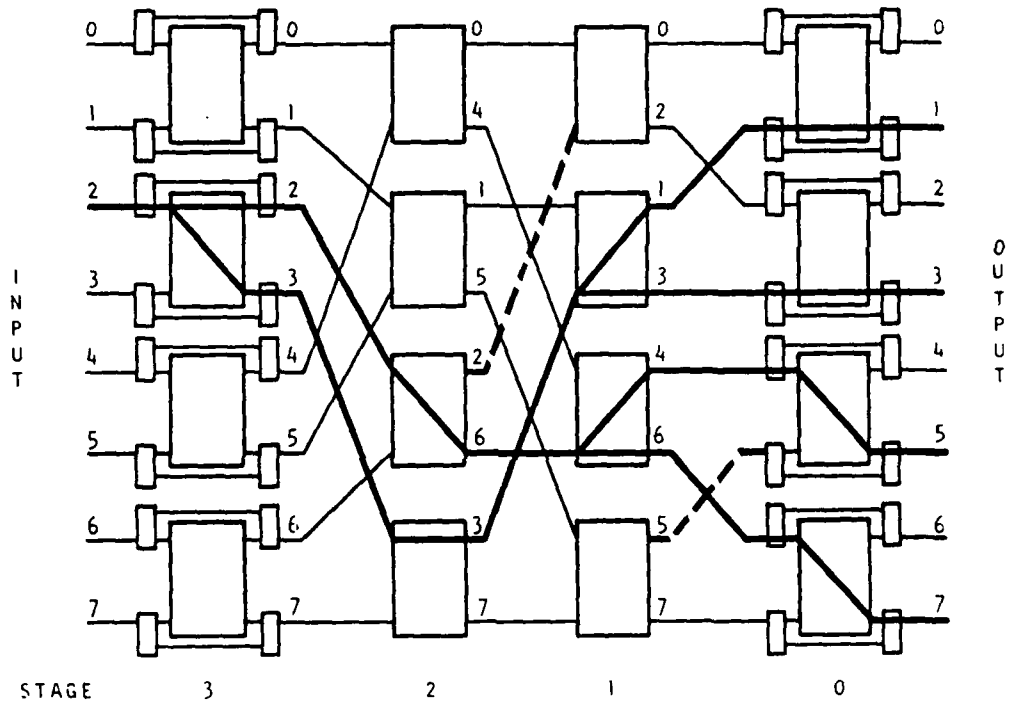


Figure 4.8 ESC network with $N=8$ and faults (2,2) and (1,5) showing a fault-free broadcast path from input 2 to outputs 1, 3, 5, and 7 that combines portions of the primary and secondary broadcast paths.

Table 4.1 summarizes the information on the consequences of, or action required in response to, multiple faults in the ESC. A set of five fault categories can describe any instance of multiple faults in an ESC network. The categories are: all faults in stage n boxes, all faults in stage 0 boxes, at least one but not all faults in stage n boxes, at least one but not all faults in stage 0 boxes, and all faults in neither stage n nor stage 0 boxes. All but two of these categories are mutually exclusive; the category "at least one but not all faults in stage n boxes" can refer to a multiple fault situation that fits "at least one but not all faults in stage 0 boxes." For each fault category the table either states whether fault-free interconnection capability is present or absent, or lists what further action is needed to make that determination.

4.5 Routing Tags

The use of routing tags to control the Generalized Cube topology has been discussed in [Law75, SiM81b]. A broadcast routing tag has also been developed [SiM81b, Wen76]. The details of one routing tag scheme are summarized here to provide a basis for describing routing tags for the ESC.

For one-to-one connections in the Generalized Cube an n -bit tag is computed from the source address, S , and the destination address, D . The routing tag $T = S \oplus D$, where \oplus means bitwise exclusive-or [SiM81b]. Let $t_{n-1} \dots t_1 t_0$ be the binary representation of T . To determine its required setting, an interchange box at stage i need only examine t_i . If $t_i = 0$, the straight state is used; if $t_i = 1$, an exchange is performed. For example, given $S = 001$ and $D = 100$, then $T = 101$, and the box settings are exchange, straight, and exchange. Figure 4.9 illustrates this route in a fault-free ESC (which effectively has the topology of the Generalized Cube network).

Table 4.1 Summary of information on the multiple fault tolerance of the ESC network. "FFIC" stands for fault-free interconnection capability.

<u>Fault Category</u>	<u>Consequence or Action Required</u>
All faults in stage n boxes	FFIC retained
All faults in stage 0 boxes	FFIC retained
At least one but not all faults in stage n boxes	FFIC lost
At least one but not all faults in stage 0 boxes	FFIC lost
All faults in network components other than stage n or stage 0 boxes	Test all faults labels using either Theorem 4.6 or Corollary 4.1 for one-to-one connections, or Corollary 4.2 for broadcast connections

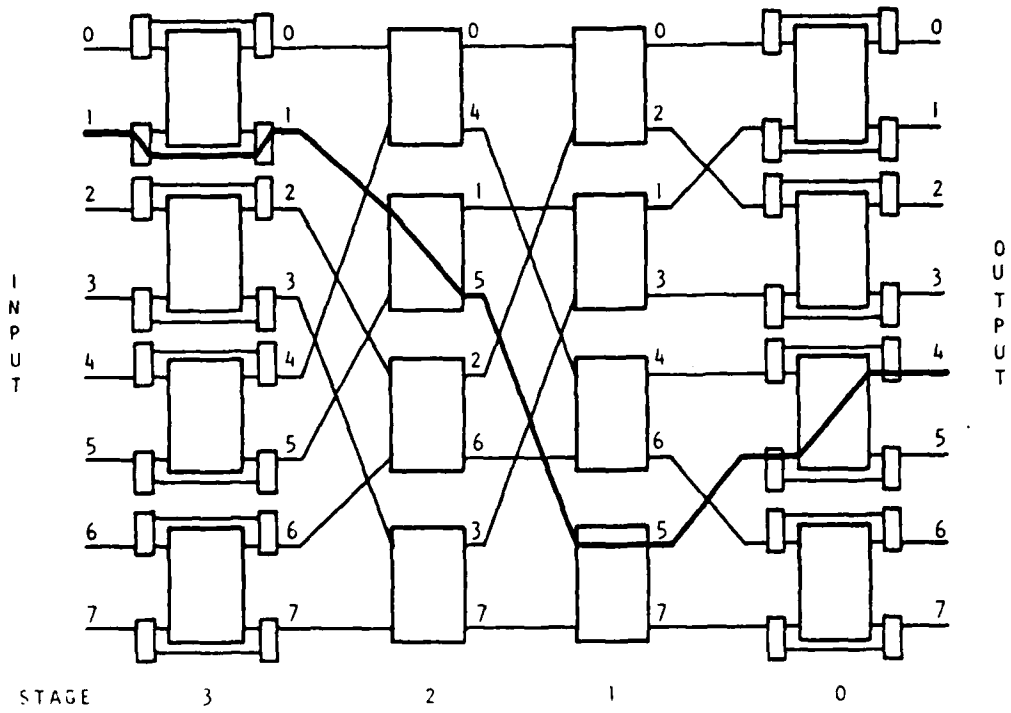


Figure 4.9 Path used when routing from input 1 to output 4 in a fault-free ESC with $N = 8$.

The routing tag scheme can be extended to allow broadcasting from a source to a power of two destinations with one constraint. That is, if there are 2^j destinations, $0 < j \leq n$, then the Hamming distance (number of differing bit positions) [Lin70] between any two destination addresses must be less than or equal to j [SiM81b]. Thus, there is a fixed set of j bit positions where any pair of destination addresses may disagree, and $n-j$ positions where all agree. For example, the set of addresses $\{010, 011, 110, 111\}$ meets the criterion.

To demonstrate how a broadcast routing tag is constructed, let S be the source address and D^1, D^2, \dots, D^{2^j} be the 2^j destination addresses. The routing tags are $T_i = S \oplus D^i$, $1 \leq i \leq 2^j$. These tags will differ from each other only in the same j bit positions in which S may differ from D^i , $0 < i \leq 2^j$.

The broadcast routing tag must provide information for routing and determining branching points. Such tags consist of two parts, each with n bits, for a total of $2n$ bits. Let the routing information be $R = r_{n-1} \dots r_1 r_0$ and the broadcast information be $B = b_{n-1} \dots b_1 b_0$. The j bits where tags T_i differ determine the stages in which broadcast connections will be needed. The broadcast routing tag $\{R, B\}$ is constructed by setting $R = T_i$ for any i , and $B = D^k \oplus D^l$, where D^k and D^l , where are any two destinations which differ by j bits.

To interpret $\{R, B\}$, an interchange box in stage i must examine r_i and b_i . If $b_i = 0$, r_i has the same effect as t_i , the i^{th} bit of the one-to-one connection tag. If $b_i = 1$, r_i is ignored and an upper or lower broadcast is performed depending upon whether the route uses the upper or lower box input, respectively. For example, if $S = 101$, $D^1 = 010$, $D^2 = 011$, $D^3 = 110$, and $D^4 = 111$, then $R = 111$ and $B = 101$. The network configuration for this broadcast is shown in Figure 4.10 for a fault-free ESC.

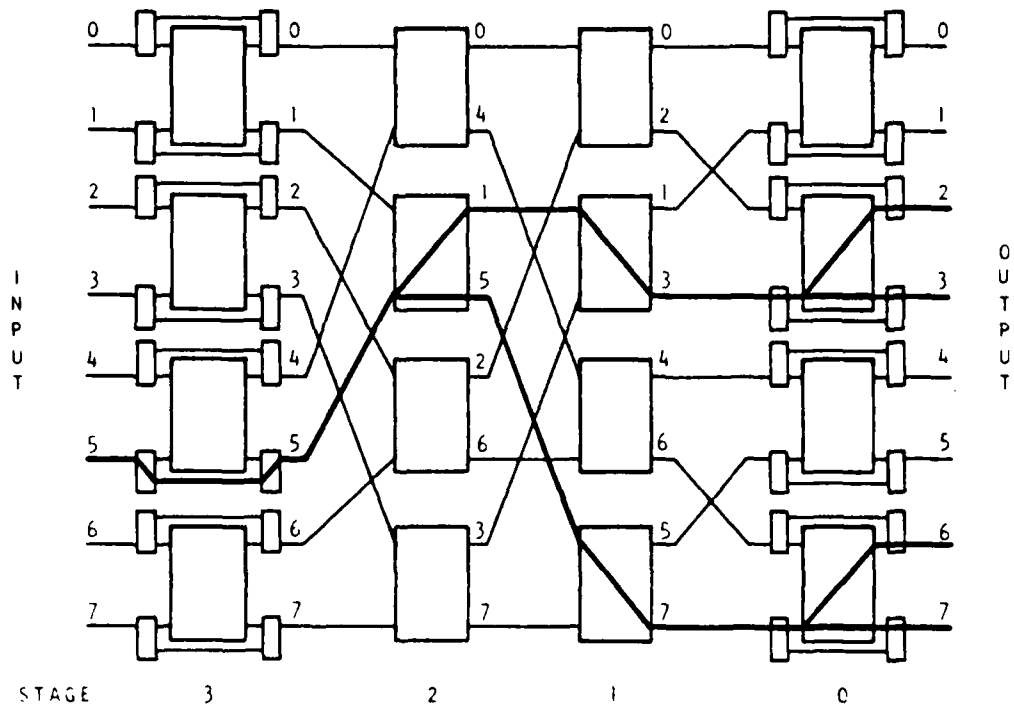


Figure 4.10 Broadcast path for broadcasting from input 5 to outputs 2, 3, 6, and 7 in a fault-free ESC with $N = 8$.

Both routing tags and broadcast routing tags for the ESC that take full advantage of its fault tolerant capabilities can be derived from the tag schemes for the Generalized Cube. The ESC uses $n = 1$ bit routing tags. The one-to-one routing tag is $T' = t'_n \dots t'_1 t'_0$ and the broadcast routing tag is $\{R', B'\}$, where $R' = r'_n \dots r'_1 r'_0$ and $B' = b'_n \dots b'_1 b'_0$. The additional bit position is to control stage n . Actual tag values depend on whether the ESC has a fault as well as on the source and destination addresses, but are readily computed.

First consider the fault-free case. For both routing and broadcast tags, the n^{th} bit will be ignored since stage n is disabled when there are no faults. The routing tag is given by $T' = t'_n t_{n-1} \dots t_1 t_0$, where $t_{n-1} \dots t_1 t_0 = T$, the tag used in the Generalized Cube. The bit t'_n can be set to either 0 or 1. The bits of T' are interpreted in the same way as tag bits in the Generalized Cube scheme. The broadcast routing tag is composed of $R' = r'_n r_{n-1} \dots r_1 r_0$ and $B' = b'_n b_{n-1} \dots b_1 b_0$, where $r_{n-1} \dots r_1 r_0 = R$, $b_{n-1} \dots b_1 b_0 = B$, and r'_n and b'_n are arbitrary. Again, the bits of $\{R', B'\}$ have the same meaning as in the Generalized Cube.

Now routing tag and broadcast routing tag definitions for use in the ESC with a fault will be described. With regard to routing tags, the primary path in the ESC is that corresponding to the tag $T' = 0t_{n-1} \dots t_1 t_0$, and the secondary path is that associated with $T' = 1t_{n-1} \dots t_1 \bar{t}_0$. The primary broadcast path is specified by $R' = 0r_{n-1} \dots r_1 r_0$ and $B' = 0b_{n-1} \dots b_1 b_0$, whereas $R' = 1r_{n-1} \dots r_1 \bar{r}_0$ and $B' = 0b_{n-1} \dots b_1 b_0$ denote the secondary broadcast path.

It is assumed that the system has appropriately reconfigured the network and distributed fault labels to all sources as required. With the condition of

the primary path known, a routing tag that avoids the network fault can be computed.

Theorem 4.7: For the ESC with one fault, any one-to-one connection performable on the Generalized Cube with the routing tag T can be performed using the routing tag T' obtained from the following rules.

1. If the fault is in stage 0, use $T' = t_0 t_{n-1} \dots t_1 t'_0$, where t'_0 is arbitrary.
2. If the fault is in a link or in a box in stages $n-1$ to 1 and the primary path is fault-free, use $T' = 0 t_{n-1} \dots t_1 t_0$. If the primary path is faulty, use the secondary path $T' = 1 t_{n-1} \dots t_1 \bar{t}_0$.
3. If the fault is in a stage n box, use $T' = t'_n t_{n-1} \dots t_1 t_0$, where t'_n is arbitrary.

Proof: Assume that the fault is in stage 0, i.e., stage n is enabled and stage 0 disabled. Since stage n duplicates stage 0 (both perform cube_0), routing can be accomplished by substituting stage n for stage 0. The tag $T' = t_0 t_{n-1} \dots t_1 t'_0$ does this by placing a copy of t_0 in the n^{th} bit position. Stage n then performs cube_0 as necessary. Note that the low-order bit position of T' , t'_0 , will be ignored since stage 0 is disabled.

Assume the fault is in a link or in a box in stages $n-1$ to 1. T specifies the primary path. If this path is fault-free, setting $T' = 0 t_{n-1} \dots t_1 t_0$ will use this path. The 0 in the n^{th} bit position is necessary because stages n and 0 are enabled, given the assumed fault location. If the path denoted by T contains the fault, then the secondary path is fault-free by Theorem 4.2 and must be used. It is reached by setting the high-order bit of T' to 1. This maps S to the

input $\text{cube}_0(S)$ of stage $n-1$. To complete the path to D , bits $n-1$ to 0 of T' must be $\text{cube}_0(S) \oplus D = t_{n-1} \dots t_1 \bar{t}_0$. Thus, $T' = 1t_{n-1} \dots t_1 \bar{t}_0$.

Finally, assume the fault is in stage n . Stage n will be disabled, and the routing tag needed will be the same as in the fault-free ESC.

□

Recall from Section 4.3.3 that the procedure for determining if a primary broadcast path is faulty may result in unnecessary use of the secondary broadcast path. As the following theorem shows, generating broadcast routing tags to use the secondary broadcast path incurs minimal additional overhead relative to primary broadcast path tags. Thus, unnecessary use of secondary broadcast paths is of little consequence.

Theorem 4.8: For the ESC with one fault, any broadcast performable on the Generalized Cube with the broadcast routing tag $\{R, B\}$ can be performed using the broadcast routing tag $\{R', B'\}$ obtained from the following rules.

1. If the fault is in stage 0 , use $R' = r_0 r_{n-1} \dots r_1 r'_0$ and $B' = b_0 b_{n-1} \dots b_1 b'_0$, where r'_0 and b'_0 are arbitrary.
2. If the fault is in a link or in a box in stages $n-1$ to 1 and the primary broadcast path is fault-free, use $R' = 0r_{n-1} \dots r_1 r_0$ and $B' = 0b_{n-1} \dots b_1 b_0$. If the secondary broadcast path has been chosen, use $R' = 1r_{n-1} \dots r_1 \bar{r}_0$ and $B' = 0b_{n-1} \dots b_1 b_0$.
3. If the fault is in a stage n box, use $R' = r'_n r_{n-1} \dots r_1 r_0$ and $B' = b'_n b_{n-1} \dots b_1 b_0$, where r'_n and b'_n are arbitrary.

Proof: Assume that the fault is in a stage 0 box. As a direct consequence of Lemma 4.5, any broadcast performable on the Generalized Cube using the broadcast routing tag $\{R,B\}$ is performable on the ESC with stage 0 disabled and stage n enabled (i.e., stage 0 faulty). The broadcast routing tag substitutes stage n for stage 0 by having r_0 and b_0 copied into r'_n and b'_n , respectively. This results in the same broadcast because the order in which the interconnection functions are applied is immaterial for one-to-one routing and broadcasting. Specifically, if $r_i = 0$ and $b_i = 0$, then in the set of destination addresses, $d_i = s_i$; if $r_i = 1$ and $b_i = 0$, then $d_i = \bar{s}_i$; and if $b_i = 1$, then d_i can be 1 or 0. When $b_0 = 0$ and there is a fault in stage 0, if $r_0 = 0$, the primary broadcast path is used, and if $r_0 = 1$, the secondary broadcast path is used. When $b_i = 1$ and stage 0 is faulty, the stage n interchange box routing the message performs a broadcast, and a combination of primary and secondary paths connect the source to its destinations. Each address bit is affected individually, making the order of stages irrelevant.

Assume the fault is in a link or a box in stages $n-1$ to 1. $\{R,B\}$ specifies the primary broadcast path. If it is fault-free, setting $R' = 0r_{n-1}\dots r_1r_0$ and $B' = 0b_{n-1}\dots b_1b_0$ will use this broadcast path. If the primary broadcast path contains the fault then the secondary broadcast path is fault free as a consequence of Theorem 4.4. Setting $R' = 1r_{n-1}\dots r_1\bar{r}_0$ and $B' = 0b_{n-1}\dots b_1b_0$ causes the broadcast to be performed using the secondary broadcast path.

Finally, assume the fault is in stage n . Stage n will be disabled, and the broadcast routing tag needed will be the same as in the fault-free ESC.

□

Theorems 4.7 and 4.8 are important to MIMD operation of the network because they show that the fault tolerant capability of the ESC is available through simple manipulation of the usual routing or broadcast tags. Table 4.2 summarizes routing tags and Table 4.3 summarizes broadcast routing tags for the ESC.

In the case of multiple faults where the conditions of Theorem 4.6 are met (i.e., there exists at least one fault-free path between any source and destination), routing tag utility is unchanged. That is, each source checks the primary path for faults, but against a longer list of fault labels. The routing tag is still formed as in rule 2 of Theorem 4.7.

Broadcast tags can be used to determine if the primary or secondary broadcast path of the broadcast specified by the tag contains a fault. To check if a fault in stage i is in the primary broadcast path, the source constructs $H = h_{n-1} \dots h_1 h_0$ such that for $0 \leq j < i$, $h_j = s_j$, and for $i \leq j \leq n-1$, if $b_j = 1$ then $h_j = x$ ("don't care"), otherwise $h_j = s_j \oplus r_j$. If H matches a fault label (with "x" matching 0 or 1), then the primary path contains a fault. Note that if \bar{s}_0 is used in place of s_0 , the secondary broadcast path can be checked. This test can be used for both single faults and multiple faults (by repeating the test for each fault label).

4.6 Partitioning

The *partitionability* of a network is the ability to divide the network into independent subnetworks of different sizes [Sie80]. Each subnetwork of size $N' < N$ must have all the interconnection capabilities of a complete network of that same type built to be of size N' . A partitionable network can be characterized by any limitations on the way in which it can be subdivided.

Table 4.2 One-to-one routing tags for the ESC. The symbol "x" represents either 0 or 1.

Fault Location	Routing Tag T'
No Fault	$T' = xt_{n-1} \dots t_1 t_0$
Stage 0	$T' = t_0 t_{n-1} \dots t_1 x$
Stage i box, $0 < i < n$, or any link	$T' = 0t_{n-1} \dots t_1 t_0$ if primary path is fault free; $T' = 1t_{n-1} \dots t_1 \bar{t}_0$ if primary path contains a fault
Stage n box	$T' = xt_{n-1} \dots t_1 t_0$

Table 4.3 Broadcast routing tags for the ESC. The symbol "x" represents either 0 or 1.

Fault Location	Broadcast Routing Tag $\{R', B'\}$
No Fault	$R' = xr_{n-1}\dots r_1r_0$ $B' = xb_{n-1}\dots b_1b_0$
Stage 0	$R' = r_0r_{n-1}\dots r_1x$ $B' = b_0b_{n-1}\dots b_1x$
Stage i box, $0 < i < n$, or any link	$R' = 0r_{n-1}\dots r_1r_0$ $B' = 0b_{n-1}\dots b_1b_0$ if primary broadcast path is fault free; $R' = 1r_{n-1}\dots r_1\bar{r}_0$ $B' = 0b_{n-1}\dots b_1b_0$ if primary broadcast path contains a fault
Stage n box	$R' = xr_{n-1}\dots r_1r_0$ $B' = xb_{n-1}\dots b_1b_0$

Such a network allows an MSIMD, partitionable SIMD/MIMD, or MIMD machine to be dynamically reconfigured into independent subsystems.

The Generalized Cube can be partitioned into two subnetworks of size $N/2$ by forcing all interchange boxes to the straight state in any one stage [Sie80]. All the input and output port addresses of a subnetwork will agree in the i^{th} bit position if the stage that is set to all straight is the i^{th} stage. For example, Figure 4.11 shows how a Generalized Cube with $N=8$ can be partitioned on stage 2, or the high-order bit position, into two subnetworks with $N'=4$. The two subnetworks are denoted by the A and B labels on the interchange boxes. Since both subnetworks have all the properties of a Generalized Cube, they can be further subdivided independently. This allows the network to be partitioned into various network port groupings each with a power of two ports. For example, a network of size $N=64$ could be partitioned into five subnetworks with one each of sizes 32, 16, 8, 4, and 4.

The ESC can be partitioned in a similar manner, with the property that each subnetwork has the attributes of the ESC, including fault tolerance. The only constraint is that the partitioning cannot be done using stage n or stage 0.

Theorem 4.9: The ESC can be partitioned with respect to any stage except stages n and 0.

Proof: The cube functions cube_{n-1} through cube_1 each occur once in the ESC. Setting stage i , $1 \leq i \leq n-1$, to all straight separates the network input and output ports into two independent groups. Each group contains ports whose addresses agree in the i^{th} bit position, i.e., all addresses have their i^{th} bits equal to 0 in one group, and 1 in the other. The other n stages provide the cube,

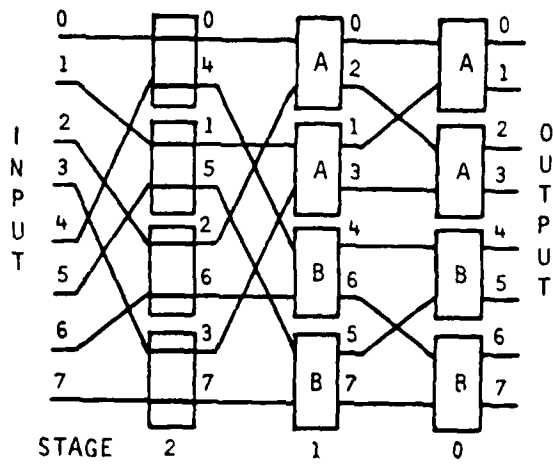


Figure 4.11 Generalized Cube network with $N=8$ partitioned into two subnetworks of size $N'=4$ based on the high-order bit position. The A and B labels denote the two subnetworks.

functions for $0 \leq j < n$ and $j \neq i$, where cube_0 appears twice. This comprises an ESC network for the $N/2$ ports of each group. As with the Generalized Cube, each subnetwork can be further subdivided. Since the addresses of the interchange box outputs and links of a primary path and a secondary path differ only in the 0^{th} bit position, both paths will be in the same partition (i.e., they will agree in the bit position(s) upon which the partitioning is based). Thus, the fault-tolerant routing scheme of the ESC is compatible with network partitioning.

If partitioning is attempted on stage n the result will clearly be a Generalized Cube topology of size N . Attempting to partition on stage 0 again yields a network of size N , in particular a Generalized Cube with cube_0 first, not last. In neither case are independent subnetworks formed.

□

In Figure 4.12 an ESC with $N=8$ is shown partitioned with respect to stage 2. The two subnetworks are indicated by the labels A and B. Subnetwork A consists of input and output ports 0, 1, 2, and 3. These port addresses agree in the high-order bit position (it is 0). Subnetwork B contains input and output ports 4, 5, 6, and 7, all of which agree in the high-order bit position (it is 1).

Partitioning can be readily accomplished by combining routing tags with masking. An $(n+1)$ -bit mask can be used to define which stages are to be used to establish a partition. By logically ANDing tags with masks to force to 0 those tag positions corresponding to interchange boxes that should be set to straight, partitions can be enforced. This process is external to the network

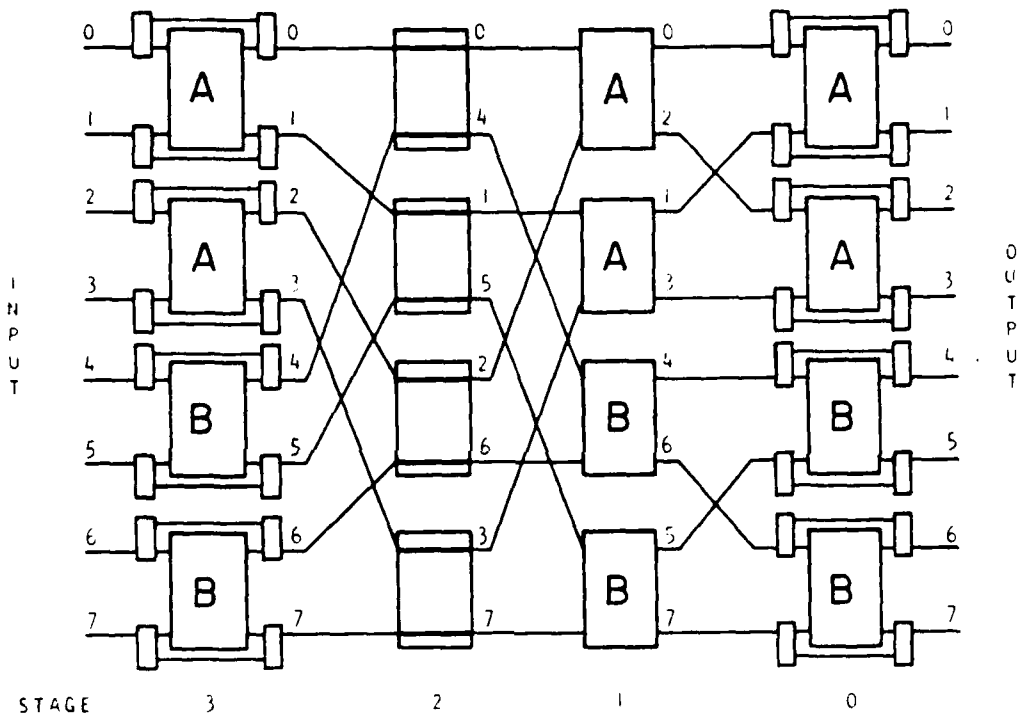


Figure 4.12 ESC network with $N=8$ partitioned into two subnetworks of size $N'=4$ based on the high-order bit position. The A and B labels denote the two subnetworks.

and, so, independent of a network fault. Thus, partitioning is unimpeded by a fault.

In PASM partitioning is designed to be based on input/output port addresses within a group agreeing in some number of low-order bit positions. Figure 4.13 shows the Generalized Cube establishing a partition on the low-order bit position. The two subnetworks are again indicated by the labels A and B. The ESC as previously defined cannot support this type of partition. However, a variation of the ESC can perform low-order bit partitioning. An ESC-like network can be constructed by adding an extra stage to the output side of the Generalized Cube network that implements cube_{n-1} . Call this new stage -1 . Thus, from the input to the output, the stages implement cube_{n-1} , cube_{n-2} , ..., cube_1 , cube_0 , and cube_{n-1} . The same fault-tolerant capabilities are available with this new network, but partitioning may be done on stage 0. Hence, low-order bit partitioning is available. Partitioning on stages $n-1$ and -1 is not available. Figure 4.14 illustrates this variation on the ESC. The two subnetworks formed by partitioning on stage 0 are labeled A and B, as before.

4.7 Permuting

In SIMD mode generally all or most sources will be sending data simultaneously. Sending data from each source to a single, distinct destination is referred to as *permuting* data from input to output. A network can *perform* or *pass* a permutation if it can map each source to its destination without conflict. *Conflict* occurs when two or more paths include the same output of a switching element.

The fault-free ESC clearly has the same permuting capability as the Generalized Cube. That is, any permutation performable by the Generalized

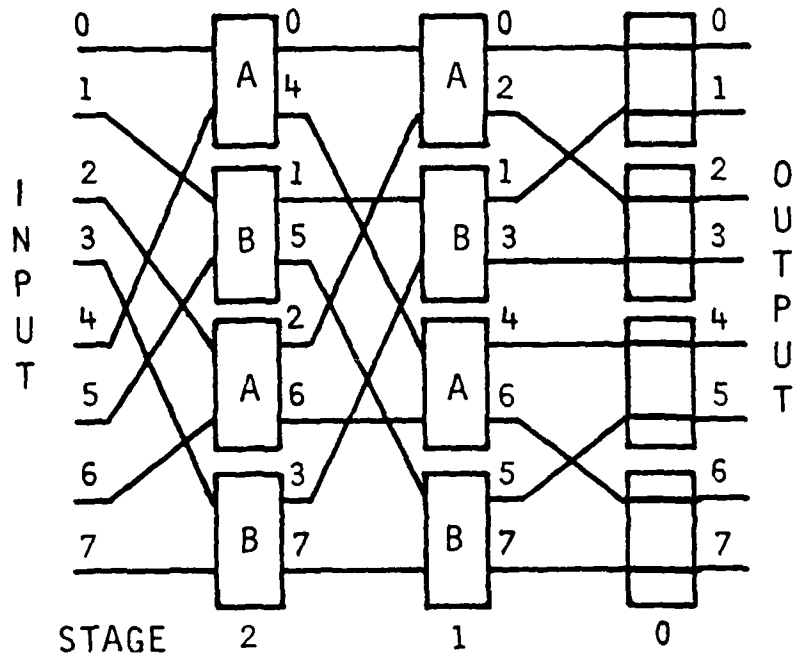


Figure 4.13 Generalized Cube network with $N=8$ partitioned into two subnetworks of size $N'=4$ based on the low-order bit position. The A and B labels denote the two subnetworks.

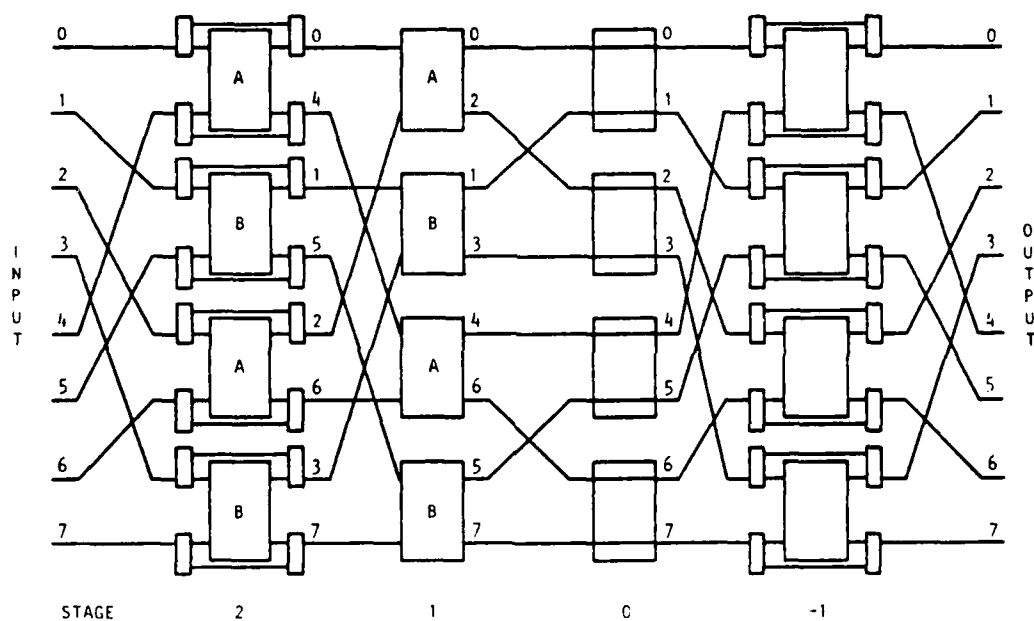


Figure 4.14 A variation on the ESC network, shown for $N=8$, that allows partitioning into two subnetworks of size $N'=4$ based on the low-order bit position. The A and B labels denote the two subnetworks.

Cube is performable by the ESC. If stage n in a fault-free ESC is enabled, the permuting capability is a superset of the Generalized Cube. Also, the ESC routing tags discussed in Section 4.5 are entirely suitable for use in an SIMD environment.

Because of its fault-tolerant nature, it is possible to perform permutations on the ESC with a single fault. It can be shown that in this situation two passes are sufficient to realize any Generalized Cube performable permutation.

Theorem 4.10: In the ESC with one fault all Generalized Cube performable permutations can be performed in at most two passes.

Proof: If a stage n interchange box is faulty, the stage is bypassed and the remainder of the ESC performs any passable permutation with a single pass. If the fault is in a stage 0 box the permutation can be accomplished in two passes as follows. In the first pass, stages n and 0 are bypassed and the remaining stages are set as usual. On the second pass, stage n is set as stage 0 would have been, stages $n-1$ through 1 are set to straight, and stage 0 is again bypassed. This simulates a pass through a fault-free network.

While stages n to 1 of the ESC provide the complete set of cube interconnection functions found in the Generalized Cube, a single pass through the stages in this order does not duplicate its permuting capability. For example, the Generalized Cube can perform a permutation which includes the mappings 0 to 0 and 1 to 2. Stages n to 1 of the ESC cannot do this. The order of the stages is important. Thus, the two pass procedure given is necessary.

When the fault is in a link or a box in stages $n-1$ to 1 , then at the stage containing the fault there are less than N paths through the network. Thus, N paths cannot exist simultaneously. The permutation can be completed in two passes in the following way. First, all sources with fault-free primary paths to their destination are routed. One source will not be routed if the failure was in a link, two if in a box. With a failed link, the second pass routes the remaining source to its destination using its fault-free secondary path. With a faulty box, the secondary paths of the two remaining sources will route to their destinations without conflict. Recall that paths conflict when they include the same box output, and the primary paths of a passable permutation do not conflict. Thus, the stage i output labels of the two primary paths are distinct, for $0 \leq i \leq n$. The secondary path stage i output labels differ from the primary path labels only by complementing the 0^{th} bit position. Therefore, the secondary path output labels are also distinct, for $0 \leq i \leq n$. Hence, the secondary paths do not conflict.

□

Permutation passing can be extended naturally to the multiple fault situation.

Corollary 4.3: In the ESC with multiple faults but retaining fault-free interconnection capability, all Generalized Cube performable permutations can be performed in at most two passes.

Proof: For a performable permutation the primary paths between each source/destination pair are by definition pairwise nonconflicting. From the

proof of Theorem 4.10, if two primary paths do not conflict then their two associated secondary paths do not conflict. Thus, there is no conflict among the secondary paths. Therefore, in the ESC with multiple faults but retaining fault-free interconnection capability, a permutation can be performed by first passing data over those primary paths which are fault-free and then passing the remaining data using secondary paths.

For multiple faults in stage n , that stage is disabled and permutations are performed in one pass. With multiple faults in stage 0 the same procedure for the case of a single stage 0 fault is used, performing permutations in two passes.

□

Figures 4.15 and 4.16 illustrate how the permutation that sends data from every source x to destination $(x + 2)$ modulo N , for $0 \leq x < N$ and $N = 8$ is performed in the presence of two faults with fault labels (2,2) and (1,4). In Figure 4.15 the fault-free primary paths that comprise the first pass are shown. The remaining fault-free secondary paths used in the second pass are indicated in Figure 4.16.

4.8 Conclusions

This chapter has presented the Extra Stage Cube interconnection network, a derivative of the Generalized Cube network. The ESC was shown to be single-fault tolerant. Under multiple faults the ESC was shown to be robust, often retaining fault-free interconnection capability. A minor adaptation of the "exclusive-or" routing tag and broadcast routing tag schemes designed for the Generalized Cube was described. This allows the use of tags to control a

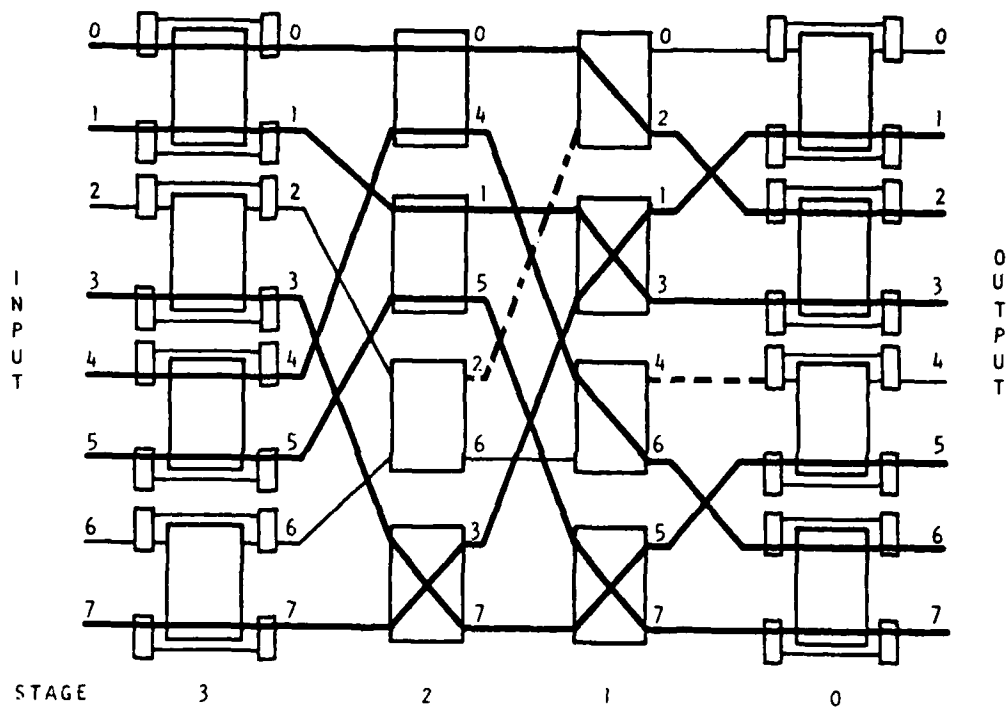


Figure 4.15 ESC network with $N=8$ and faults (2,2) and (1,4) (indicated by broken lines) showing all fault-free primary paths for the permutation mapping input x to output $(x+2) \pmod{N}$, for $0 \leq x < N$.

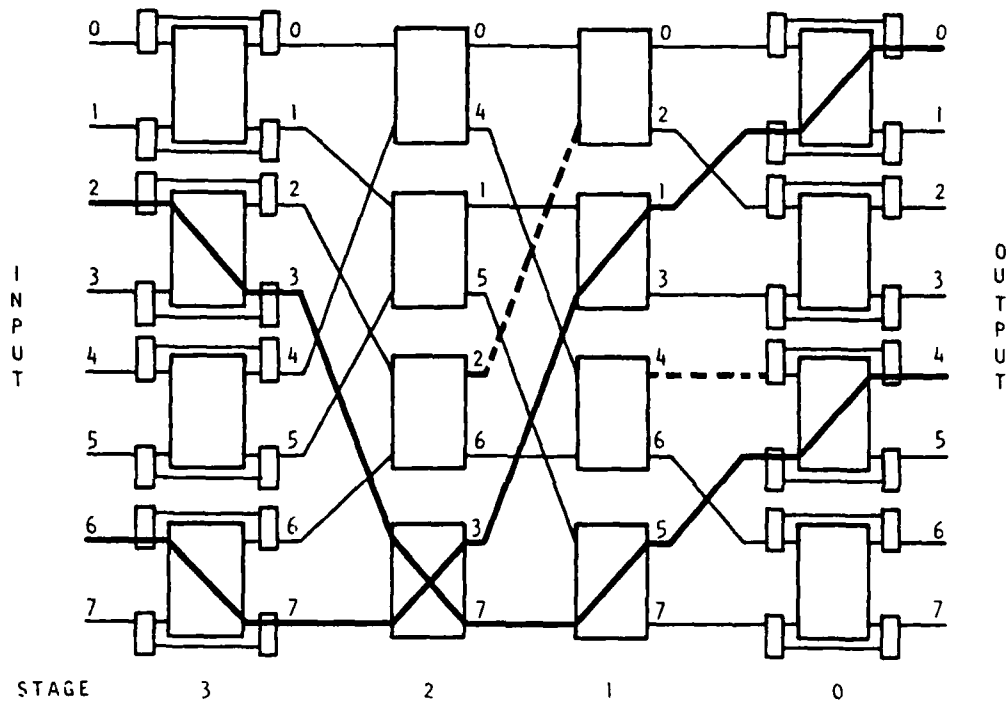


Figure 4.16 ESC network with $N = 8$ and faults (2,2) and (1,4) (indicated by broken lines) showing the secondary paths corresponding to faulty primary paths for the permutation mapping input x to output $(x + 2)$ modulo N , for $0 \leq x < N$.

faulted, as well as fault-free, ESC. The partitioning and permuting abilities of the ESC were discussed.

The reliability of large-scale multiprocessor systems is a function of system structure and the fault tolerance of system components. Fault-tolerant interconnection networks can aid in achieving satisfactory system reliability. The ESC seems to be a practical and useful interconnection network for such systems.

The family of multistage interconnection networks of which the Generalized Cube is representative has received much attention in the literature. Because of its relationship to the Generalized Cube network (which is representative of many multistage interconnection networks), realistic fault model and fault-tolerance criterion, and ease of routing in the presence of faults, the ESC network may also be useful as a standard for evaluating existing and future fault-tolerant networks.

CHAPTER 5
SURVEY OF FAULT-TOLERANT MULTISTAGE
INTERCONNECTION NETWORKS AND
COMPARISON TO THE EXTRA STAGE CUBE

5.1 Introduction

Many interconnection networks have been proposed in the literature. Of these, multistage networks compose a large subset. Of multistage networks, a somewhat smaller group have an ability to tolerate at least some component failures. In this latter group are those multistage interconnection networks for which the topology, in combination with switching element structure, has been designed specifically to achieve some particular fault-tolerance capability (typically, single-fault tolerance). It is these interconnection networks on which this chapter is focused.

There are a number of ways to realize a fault-tolerant interconnection network that do not involve design of the network *per se*. One way is to use error correcting codes in the data and control paths of a network not otherwise fault tolerant [LLY82]. Another scheme is to implement the network in several independent bit-slices; failure in one or more slices can be tolerated by using the remaining good slices (e.g., [SiM81b]). Yet another method is to replicate an existing network and utilize the copies in parallel, perhaps cross-linking them, so that a failure in the set does not compromise the functionality of the whole (e.g., [KrM83, ReK84]). Often these techniques, and others, are used in

conjunction with a basic fault-tolerant design.

Multistage interconnection networks for parallel processing designed specifically for fault tolerance have become an active area of research recently, and survey papers are few [AgK83, AdS84a]. This chapter presents the Modified Baseline network [WFL82], the Augmented Delta network [DiJ82], the Multipath Omega network [PaL83a], the F-network [CiS82], the Enhanced Inverse Augmented Data Manipulator [McS82a], the Gamma network [PaR82, PaR84], the Fault-Tolerant Beneš network [Agr82, SoR80], the Augmented C-network [ReK84], and β -networks [ShH84]. The papers introducing all but two of these networks (the Fault-Tolerant Beneš network and β -networks) appeared in the literature subsequent to papers describing the ESC [AdS82a, AdS82c, AdS82d]. For this reason, this chapter appears after the development of ESC properties contained in Chapter 4.

Following the survey, these networks are compared to the ESC on the basis of fault models and fault-tolerance criteria. Then a common fault-tolerance model is assumed and the fault tolerance of each network is evaluated in that circumstance. The ESC fault-tolerance model is chosen as the common model for this phase of the comparison because it is the most demanding of those of the various networks. The intent of this chapter is to provide a context in which to view the merits of the ESC fault-tolerant multistage interconnection network investigated in this research.

5.2 Survey

The surveyed networks fall into four general categories. The Modified Baseline, Augmented Delta, Multipath Omega, and F-network form a group based on the Generalized Cube network [SiM81b, SiS78]. The first two of these achieve fault tolerance by adding an extra stage of switches to a basic network which is isomorphic to the Generalized Cube topology. The Multipath Omega uses an extra stage or stages of switches, or substitutes different switches in a stage or stages, or some combination of these methods, to add fault tolerance to the Omega network [Law75], which is also isomorphic to the Generalized Cube topology. The F-network gains fault tolerance by using a Generalized Cube network structure with additional links.

The Enhanced Inverse Augmented Data Manipulator (Enhanced IADM) and Gamma networks represent a second group: the data manipulator [Fen74] class of networks. The Enhanced IADM network uses additional links, and the Gamma network uses increased switching element complexity, to realize fault tolerance.

The Fault-Tolerant Beneš network is a third type of network. It uses $2n-1$ stages of switching elements with one additional switching element to provide fault tolerance, compared to the n stages of switches in a Generalized Cube, where $N = 2^n$ is the number of inputs.

The Augmented C-network and β -networks fall into a fourth network category. Each is actually a family of networks, spanning a wide range of topologies. The topology of Augmented C-networks is inherently fault tolerant, while β -networks combine topology and an operational technique to achieve fault tolerance.

5.2.1 Modified Baseline Network

The Modified Baseline network [WFL82] is derived from the Baseline multistage interconnection network [WuF80]. The Baseline topology consists, in general, of a stage of N/r r -input/ t -output ($r \times t$) crossbar switching elements connected to t subnetworks numbered 0 to $t-1$. This yields a network with N inputs and N outputs. If the t outputs of each $r \times t$ switch are numbered 0 to $t-1$, and the switches are numbered 0 to $\frac{N}{r}-1$, then output i of switch j is connected to input j of subnetwork i , for $0 \leq i \leq t-1$, and $0 \leq j \leq \frac{N}{r}-1$. Each subnetwork is structured in the same manner as the entire network. This recursive process is carried out until the subnetworks are $r \times t$ switches, which can be implemented directly.

The Baseline network has but one path between any source and destination. Thus, any network component failure affects communication for some set of inputs and outputs. To lessen this difficulty an extra stage of switching elements is added to the Baseline network. Figure 5.1 shows the Modified Baseline network for $r=t=2$ and $N=8$, and indicates the original Baseline network and the additional stage. If an extra stage incorporating switching elements with t outputs is added at the input side of the network, then there are t connection paths for any input/output pair.

The Modified Baseline network fault model states that the only network components that can fail are switching elements not in the input or output stages. Faulty switches are considered unusable. The fault-tolerance criterion is retention of full access [CiS82]. Recall that full access is the ability to connect any given input to any output. The Modified Baseline network is single-fault tolerant and robust in the presence of multiple faults with respect

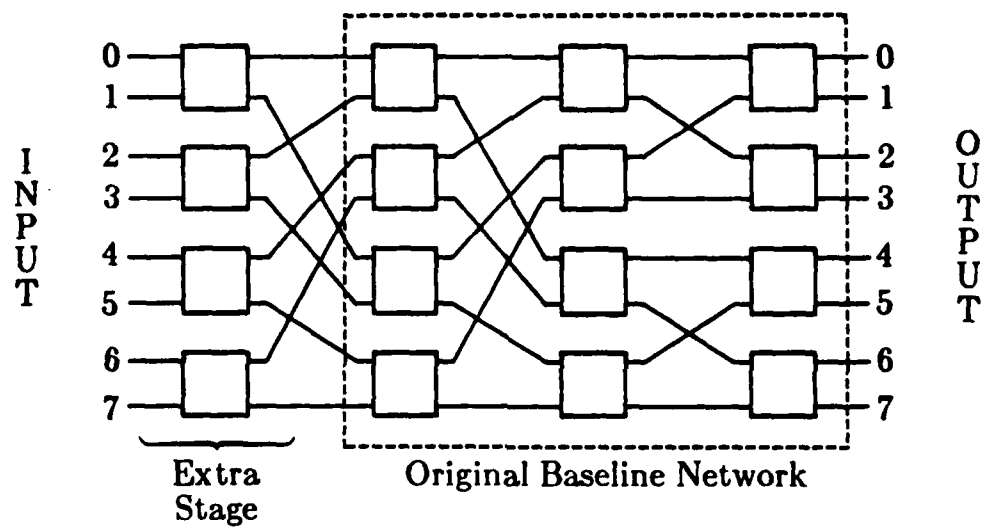


Figure 5.1 The Modified Baseline network for $N=8$.

to its fault-tolerance model and fault-tolerance criterion.

Routing in the Baseline network is carried out using *destination tags* [Law75] that consist of the address of the intended destination of a message. If $r \times r$ switches are used ($r = 2$ for Figure 5.1) then a destination address D can be represented by a base- r number $d_{m-1} \dots d_1 d_0$ where $m = \log_r N$. This base- r representation is used to select a path through the network in the following way. The switching element connected to the source will use its output numbered d_{m-1} to link to a switching element in the next stage. At stage i , d_i is used to determine the selection of switch output, $0 \leq i \leq m-1$. For the Modified Baseline network an extra digit can be appended to Baseline network destination tags to control the extra stage. This assumes that sufficient information is available to determine the value of this extra digit to avoid any existing fault [WFL82].

5.2.2 Augmented Delta Network

Delta, or *digit controlled*, networks [Pat81] are a class of multistage interconnection networks and also a subset of a very broad class known as banyan networks [GoL73]. A delta network is, in general, an $a^n \times b^n$ network with n stages, each consisting of $a \times b$ crossbar switching elements. The link pattern between stages provides a unique path of fixed length between any network input and output. Further, the link pattern is such that information can be routed from an input to an output using the switching element output corresponding to a base- b digit in the base- b representation of the output number.

An Augmented Delta network [DiJ82] can be constructed from a Delta network, as illustrated by Figure 5.2. Let $w = \log_b N$. The Augmented Delta

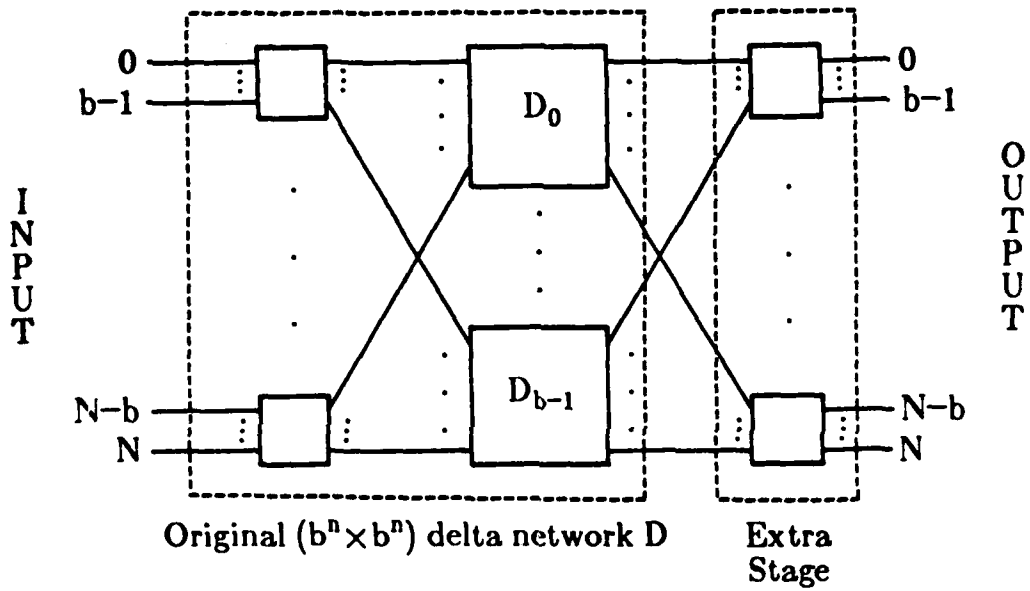


Figure 5.2 An Augmented Delta network constructed from a $b^n \times b^n$ Delta network.

network, shown in Figure 5.2, consists of a stage of N/b $b \times b$ switching elements connected to b delta networks, each of size $b^{w-1} \times b^{w-1}$. These b networks are labeled D_0 through D_{b-1} in the figure and each is structured in the same way as the $b^w \times b^w$ network. Adding a stage of N/b $b \times b$ switching elements to this, as shown in the figure, results in a topology with b paths between any pair of network input/output ports, given the switching elements are $b \times b$ devices. Additional redundant paths can be provided by adding more stages. The Augmented Delta network is similar to the Modified Baseline network; the distinction between the two is that the definition of the Augmented Delta network allows more than one extra stage and the switches of any extra stage(s) are identical to all others in the network. So that this network will be comparable with the other networks discussed in this chapter, only one additional stage is assumed.

The Augmented Delta network fault model is the following.

1. Both switching elements and links can fail.
2. Stage 0 switching elements and stage n nodes are always fault-free.
3. Faults occur independently.
4. Faulty links or switching elements are unusable.

The fault-tolerance criterion is retention of full access capability. Under this fault-tolerance model, an Augmented Delta network constructed from 2×2 switches is single fault tolerant. If $b \times b$ switching elements are used throughout, the network is $(b-1)$ -fault tolerant.

Consider packet-switched operation of the network. Let *throughput* be the average rate at which packets exit the network. *Normalized throughput* is the

ratio of throughput obtained to maximum throughput possible if no conflicts occurred in the network. When fault-free, an Augmented Delta network has a slightly lower normalized throughput than a comparable Delta network [DiJ82]. For the worst case single switching element or link fault in an Augmented Delta, performance falls to about half that of the fault-free case. On the average, however, performance is only slightly degraded.

Routing in the Augmented Delta network is performed using routing tags with one additional digit (as compared to Delta network routing tags [Pat81]) to control the extra stage. A switching element is assumed able to detect a fault in the switches and links to which it is directly connected and able to pass fault information to adjacent switches [Dia81]. When a fault occurs, neighboring switches propagate notice of the fault to their neighbors, and so on. For a bidirectional network, propagation of information proceeds to both the "inputs" and "outputs." Eventually, network input and/or output switches will receive information indicating which of their outputs or inputs, respectively, is part of a path through the network leading to the fault. Routing tags are subsequently formed so as to never use this path. Note that output switches need not be informed of the fault in a unidirectional network.

5.2.3 Multipath Omega Network

The Multipath Omega network [PaL83a, PaL83b, Pad84] is derived from the Omega multistage interconnection network [Law75]. A $B^m \times B^m$ Omega network consists of m stages of $B \times B$ crossbar switching elements linked by $(B \cdot B^{m-1})$ -shuffle interconnections. An $X \cdot Y$ -shuffle permutes $X \cdot Y$ elements. Let $0 \leq I \leq X \cdot Y - 1$, where I is represented as the $x + y$ bit binary number

$I = i_{x+y-1} \dots i_1 i_0$, $x = \log_2 X$, and $y = \log_2 Y$. Then

$$X*Y\text{-shuffle}(I) = i_{y-1} i_{y-2} \dots i_1 i_0 i_{x+y-1} \dots i_{y+1} i_y,$$

i.e., I is rotated left by x bit positions. Figure 5.3 shows an Omega network for $B = 2$, $N = 8$, and $m = 3$.

An Omega network has only one path between any input and any output; it is not single-fault tolerant. To overcome this difficulty, Omega networks with multiple paths were proposed. For the Multipath Omega network the fault-tolerance criterion is full access. Its fault model is the following.

1. Both switching elements and links fail.
2. Input and output stage switching elements are always fault-free.
3. Faults occur independently.
4. Faulty components are unusable.

Figure 5.4(a) shows a Multipath Omega network for $N = 16$. Its structure is described by the pseudofactorization $\langle 4, 2, 2, 4 \rangle$ of N . A *pseudofactorization* of N is an f -tuple $\langle B_1, B_2, \dots, B_f \rangle$ of integers with $B_1 * B_2 * \dots * B_f = B$, such that B is a multiple of N . Let $B/N = R$, then an R -path Multipath Omega network corresponding to the pseudofactorization has f stages with stage i consisting of $B_i \times B_i$ crossbar switches. Links entering stage i switches implement the $\left[k_i B_i * \frac{N}{k_i B_i} \right]$ -shuffle interconnection, where $k_i \geq 1$, and k_i is an integer chosen so that there are exactly R ways to connect any network input to network any output. For the network of Figure 5.4(a), $k_1 = 1$, $k_2 = 1$, $k_3 = 8$, and $k_4 = 1$. R is known as the *redundancy* of the Multipath Omega network. For the network of Figure 5.4(a), $B/N = (4 * 2 * 2 * 4)/16 = 4$, so the

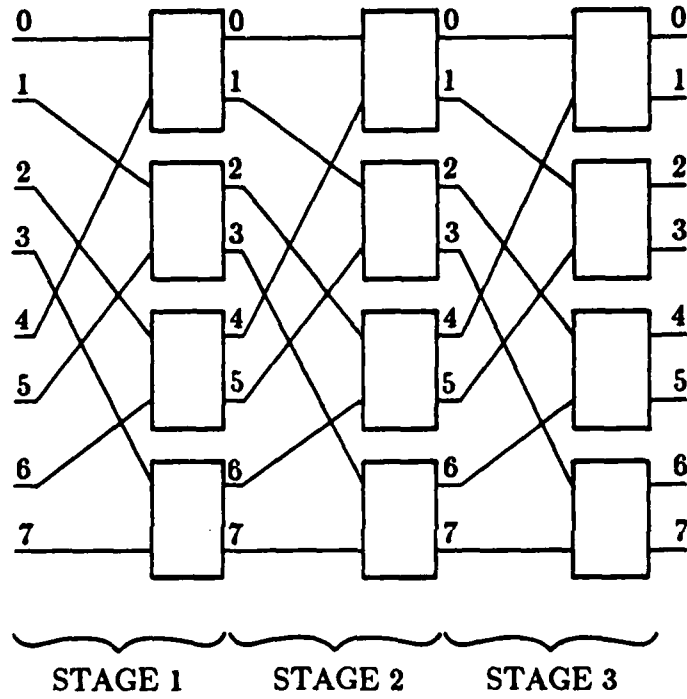
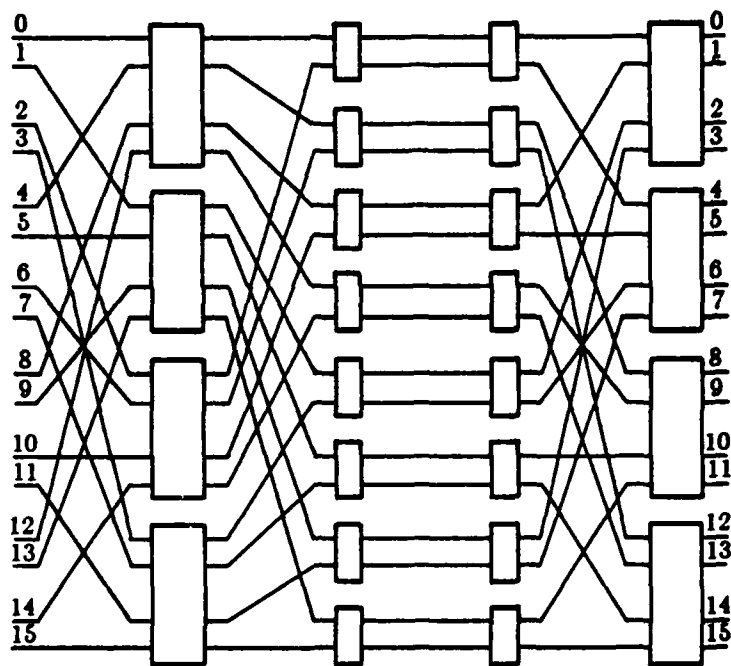
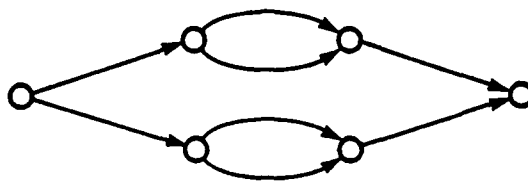


Figure 5.3 An Omega network for $B=2$, $N=8$, and $m=3$.



(a)



(b)

Figure 5.4 (a) A Multipath Omega network for $N = 16$ corresponding to the pseudofactorization $\langle 4, 2, 2, 4 \rangle$. (b) Its associated redundancy graph [PaL83a].

redundancy of this network is four. This can be illustrated by a redundancy graph, which shows the topology of the redundant paths.

Redundancy graphs are directed graphs with the following properties.

1. Graph nodes form S classes corresponding to the S stages of switches in the network.
2. Each edge in the graph connects a node in class i to one in class $i + 1$, $1 \leq i < S$.
3. All nodes of a class share the same in-degree and out-degree (numbers of entering and exiting edges).

The redundancy graph of a network is a subgraph of that network corresponding to all paths between any given input and any given output. Figure 5.4(b) shows the redundancy graph for the network of Figure 5.4(a). Figure 5.5 depicts other possible redundancy graphs, each relating to a different network structure.

The *line connectivity* of a redundancy graph is the number of distinct paths from the node in the class corresponding to the input stage of the network to the node corresponding to the output stage, i.e., the number of distinct paths between any input and any output. The redundancy graphs of Figure 5.5(a) has a line connectivity of four; that of Figure 5.5(b), two. The graph of Figure 5.5(c) has a line connectivity of one; it is the redundancy graph for an Omega network with four stages of switches. A Multipath Omega network having a redundancy graph with line connectivity λ is $(\lambda - 1)$ -fault tolerant, so a network with the redundancy graph of Figure 5.5(a) is three-fault tolerant.

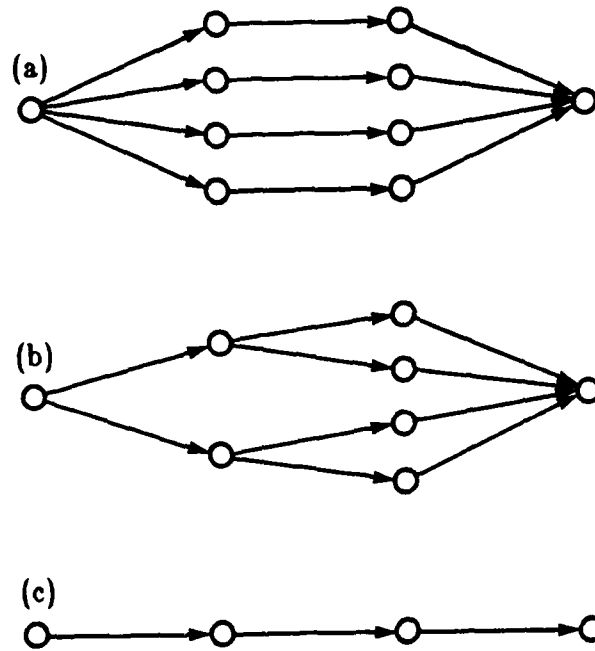


Figure 5.5 Several redundancy graphs [PaL83a].

Routing in the Multipath Omega network is controlled by routing tags and is similar to the procedure used for the Omega network [Law75]. Stage i switch settings are controlled by $b_i = \log_2 B_i$ bits. A routing tag consists of $\sum_{i=1}^f b_i$ bits, where this sum is $n + r$, $r = \log_2 R$. The destination address determines n bits of the tag; the remaining r bits select a particular path out of the R alternatives. These r bits are termed *redundant bits*.

Routing tags that specify a fault-free path are generated by one of three possible methods described in [PaL83a]. In *non-adaptive routing* a source learns of a fault only when the path it is attempting to establish reaches the faulty network element. Notice is sent back to the source, which tries the next alternative path (the redundant bits in the routing tag are changed so as to increment the binary number that can be formed by concatenating these bits). This approach requires little hardware but may have poor performance. Two forms of *adaptive routing* are proposed. With *notification on demand* a source maintains a table of faults it has encountered and uses this information to guide future routing. With *broadcast notification* of a fault, all sources that can use the faulty component in a path are notified and keep a table, as with notification on demand.

5.2.4 F-Network

The F-network [CiS82] connects $N = 2^n$ inputs to N outputs via $n + 1$ stages of N switching elements which are, in general, 4-input/4-output devices that connect one input to one output. A switching element in stage j , P_j , is

denoted by a bit string $P_j = p_{n-1} \dots p_1 p_0$. It connects to the stage $j+1$ switching elements

$$P_{j+1} = p_{n-1} \dots p_1 p_0 ,$$

$$Q_{j+1} = p_{n-1} \dots p_{j+1} \bar{p}_j p_{j-1} \dots p_1 p_0 ,$$

$$R_{j+1} = \bar{p}_{n-1} \dots \bar{p}_{j+1} \bar{p}_j p_{j-1} \dots p_1 p_0 , \text{ and}$$

$$S_{j+1} = \bar{p}_{n-1} \dots \bar{p}_{j+1} p_j p_{j-1} \dots p_1 p_0 .$$

Figure 5.6 shows the F-network for $N=8$. Stages are numbered from left to right ranging from 0 to n , and within each stage, switching elements are numbered from 0 to $N-1$. The F-network contains the structure of the Generalized Cube network and can emulate it using only the P_{j+1} and Q_{j+1} connections. Thus, the fault tolerance approach of the F-network is to add links (R_{j+1} and S_{j+1}) to the Generalized Cube structure, unlike the approach of the Modified Baseline and Augmented Delta networks.

The F-network fault model assumes the following [CiS82].

1. Only switching elements fail.
2. Stage 0 and n switching elements are always fault-free.
3. Faults occur independently.
4. A faulty switching element is unusable.

The fault-tolerance criterion for the F-network is retention of full access. The network is single fault tolerant and robust in the presence of multiple faults [CiS82] with respect to its fault-tolerance model. An expression for network *mean time before failure (MTBF)* is derived in [CiS82].

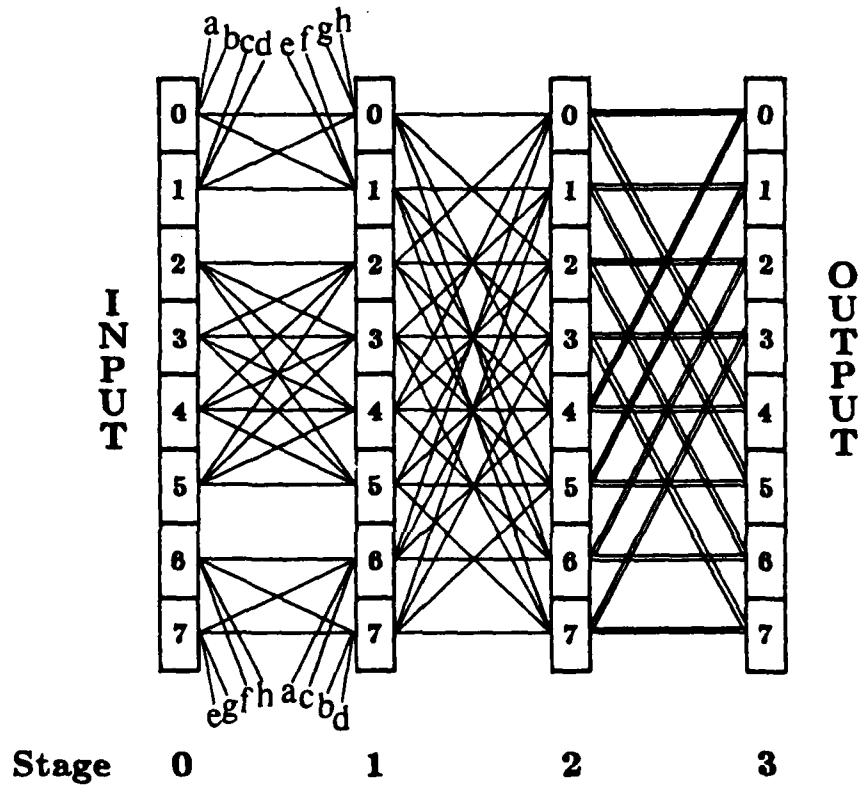


Figure 5.6 The F-network for $N=8$.

Routing in the F-network is accomplished with routing tags defined as follows [CiS82]. Let the source have address S and the destination address D . Define the routing tag $C = c_{n-1} \dots c_1 c_0 = S \oplus D$, where \oplus is the bitwise exclusive-or of the binary representations of S and D . Let r be a binary variable initially set to 0, and let r^j be the value of r after j steps of the routing algorithm in Figure 5.7.

At each step of the algorithm it is possible to calculate the choice of next stage switch in two ways. Two different switches can be selected for all stages except n , the output stage, by changing the value assigned to x . The fault tolerance of the F-network arises from this ability to choose two switching elements in the next stage to continue a path. A faulty or busy switch can be avoided by taking the appropriate path. This is done by changing the value of x upon detecting that the selected next switch is faulty or busy and recalculating the choice of next switch based on the new value of x . Because all paths from a given source go through the same stage 0 switch, and all paths to a given destination go through the same stage n switch, only interior stage switching element faults (not stage n or 0) can be tolerated.

5.2.5 Enhanced Inverse Augmented Data Manipulator Network

The Enhanced Inverse Augmented Data Manipulator network is derived from the Inverse Augmented Data Manipulator (IADM). The IADM network is an Augmented Data Manipulator (ADM) network [SiS78] with the order of stage traversal reversed. The ADM is derived from the data manipulator network [Fen74]. Figure 5.8 shows the IADM for $N=8$. It consists of $n = \log_2 N$ stages, where each stage consists of N switching elements and $3N$ links that are connected to the succeeding stage. Each switching element

```

for  $j = 0$  to  $n - 1$  begin
  assign  $x$  to be either 0 or 1;
  if  $x = 0$  then begin
    if  $c_j \oplus r^j = 0$  then
      choose stage  $j + 1$  switch  $P_{j+1}$ 
    else choose stage  $j + 1$  switch  $Q_{j+1}$ ;
     $r^{j+1} = r^j$ ;
    end
  else
    if  $2 + (c_j \oplus r^j) = 2$  then
      choose stage  $j + 1$  switch  $S_{j+1}$ 
    else choose stage  $j + 1$  switch  $R_{j+1}$ ;
     $r^{j+1} = \bar{r}^j$ ;
  end;

```

Figure 5.7 Routing algorithm for the F-network.

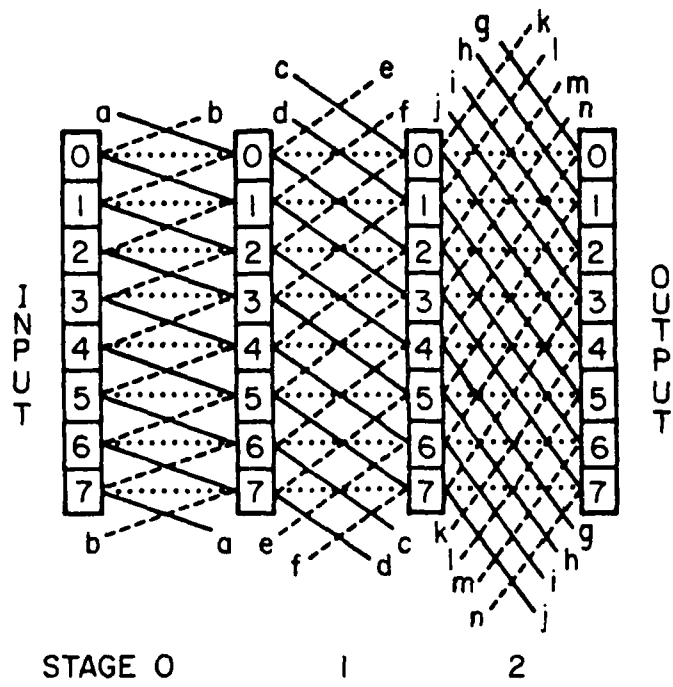


Figure 5.8 The Inverse Augmented Data Manipulator network for $N=8$.

connects one of its three inputs to one of its three outputs. At stage i , $0 \leq i < n$, the first output of switching element j , $0 \leq j < N$, is connected to an input of switching element $(j - 2^i)$ modulo N in stage $i + 1$. The second output is connected to an input of switching element j in stage $i + 1$. Finally, the third output is connected to an input of switching element $(j + 2^i)$ modulo N in stage $i + 1$. These links are known as the minus, straight, and plus links, respectively. Since $j - 2^{n-1}$ is congruent to $(j + 2^{n-1})$ modulo N , there are actually just two distinct logical data paths from each switching element in stage $n - 1$ (stage 2 in Figure 5.8). There is an additional set of N nodes at the output stage to receive data.

Performance and fault-tolerance improvements for the IADM are discussed in [McS82a]. The resulting network is called the Enhanced IADM. Its fault model is the same as for the Augmented Delta network, as is the fault-tolerance criterion. Note that the IADM is a robust network with respect to this model and criterion, but it is not single-fault tolerant.

One method of providing fault tolerance for the IADM is adding redundant straight links. A faulty straight link can be avoided by using the second straight link. Faulty plus or minus links can be avoided by taking the alternative path available at the stage just prior to the faulty link [McS82a]. However, switching element faults cannot be tolerated by adding redundant straight links.

A second, more effective modification to gain fault tolerance is to add *half links* to each of stages 1 through $n - 1$. Half links connect a switching element m in stage i to switching elements $(m + 2^{i-1})$ modulo N and $(m - 2^{i-1})$ modulo N . This is shown for $N = 8$ in Figure 5.9. Adding half links provides single-fault tolerance to any switching element or link failure. This is

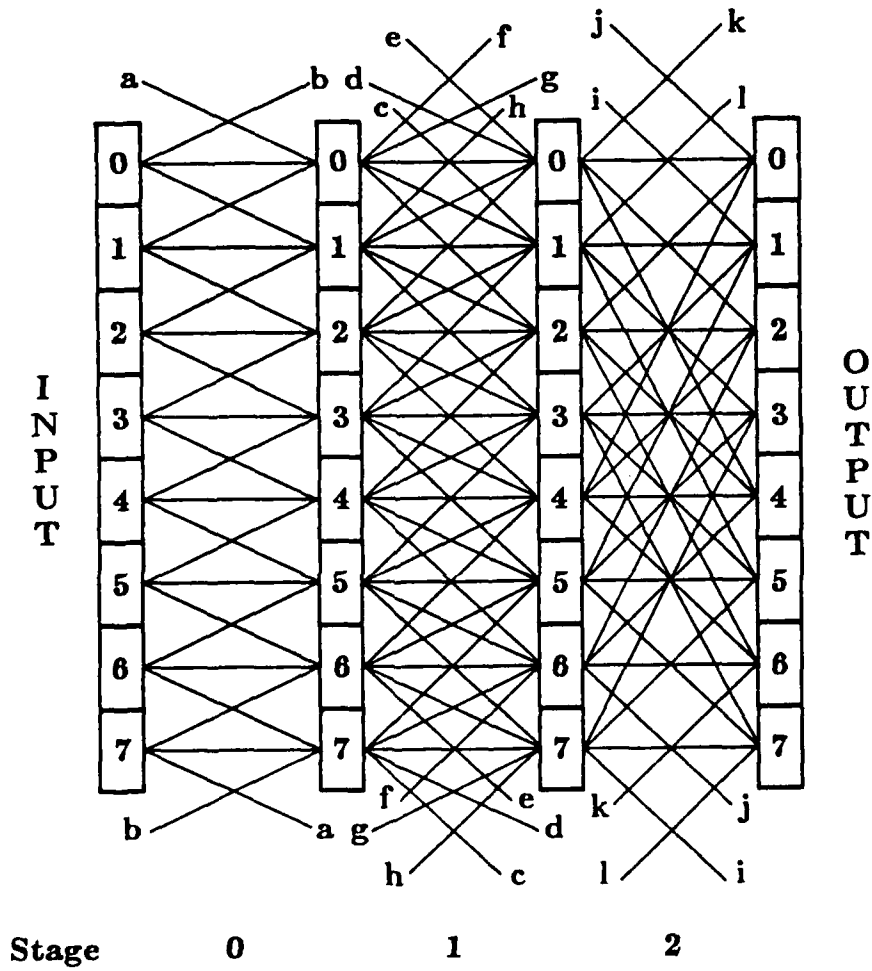


Figure 5.9 The Enhanced Inverse Augmented Data Manipulator network with half links for $N = 8$.

because at any switching element (except those in stage $n-1$, the last stage) along a route from a network input to output there are at least two (sometimes three) links leading to distinct switching elements in the successive stage, any of which can be used to satisfy the overall routing need. With a single-stage look-ahead technique [McS82a] the network becomes dynamically two-fault tolerant. That is, messages will not be sent along a route on which all alternative paths to the next stage are blocked by the two faults. Further modifications of the two hardware enhancement schemes presented are discussed in [McS82a].

Routing for the Enhanced IADM network with redundant straight links is exactly the same as for the IADM network [McS82a], because no new paths between inputs and outputs are provided by the additional links. A routing tag, T , for the IADM network (with or without redundant straight links) is computed as $T = t_n t_{n-1} \dots t_1 t_0 = D - S$, where S is the source address, and D the destination address. The tag T is expressed in signed magnitude notation, where t_n is the sign bit.

For the Enhanced IADM network with half links the routing tag scheme uses a tag T computed as $T = D - S$ if $D \geq S$, else $T = 2^n - (S - D)$. Each switching element is assumed to have the ability to determine if the switching elements or links to which it is connected are faulty and, if so, to modify routing tags dynamically to avoid a faulty component. Considerable switching element logic must be devoted to interpreting and modifying tags as information flows through the network if the full fault-tolerance capabilities of the network are to be achieved. This makes the switching elements appropriate for VLSI implementation. Note that no burden is placed on devices using the network in achieving these fault-tolerance capabilities.

5.2.6 Gamma Network

The Gamma network [PaR82, PaR84] is an adaptation of the IADM network (see Figure 5.8) and has redundant paths connecting $2^n = N$ inputs to N outputs and consists of n stages of N switches. The Gamma network is shown for $N = 8$ in Figure 5.10. However, unlike the IADM, each of the switching elements is, in general, a 3-input/3-output crossbar switch instead of a one-of-three inputs to one-of-three outputs selector. Switching elements in the input stage have only one input and three outputs, while output stage switches have three inputs and only one output. The link connection pattern is identical to that of the IADM.

The fault tolerant nature of the Gamma network can be related to certain number systems. A *number system* is a method for expressing numerical values. In a radix r number system, values are represented by digit strings where each digit can have any of the r values $\{0, 1, \dots, r-1\}$. Number systems in which digits have more than r possible values can be constructed. Let each digit be in the set $\{-a, -(a-1), \dots, -1, 0, 1, \dots, a\}$. When $r \geq 2$ and $a = r-1$ a *radix r fully redundant number system* is formed in which each digit is in the range $\{-(r-1), -(r-2), \dots, -1, 0, 1, \dots, r-1\}$. This number system is redundant in the sense that some values will have more than one representation. In fact, all non-zero values have multiple forms. A radix 2 (binary) fully redundant number system can use the digits 1, 0, and $\bar{1}$, where $\bar{1}$ corresponds to -1 .

Consider a source, S , a destination, D , and their difference $(D-S)$ modulo N . Each representation of $(D-S)$ modulo N and $\{(D-S) \text{ modulo } N\} - N$ in the binary fully redundant number system corresponds to a path in the Gamma network connecting S to D . Thus, as long

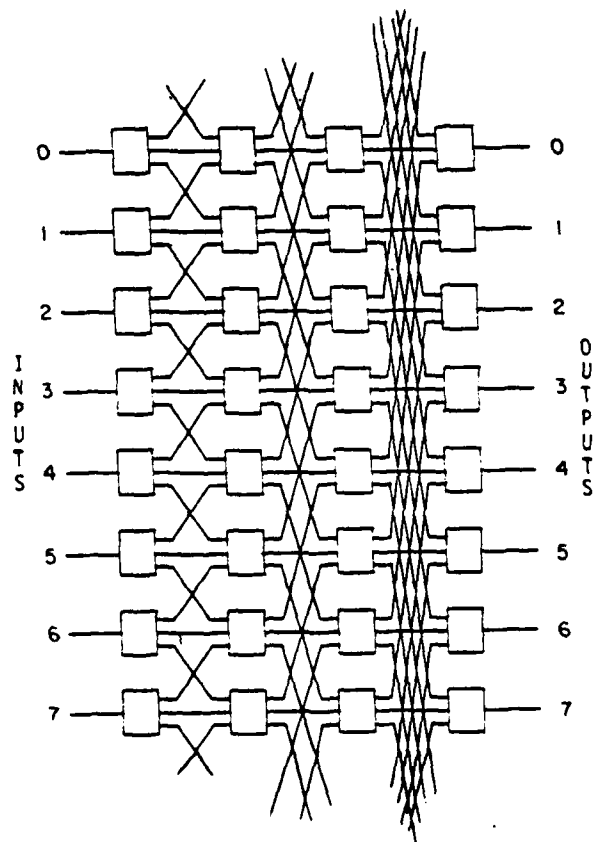


Figure 5.10 The Gamma network for $N=8$ [PaR82].

as $S \neq D$ there will be multiple paths. This is the source of Gamma network fault tolerance. If $S = D$, only one path exists (as in the IADM). In general, the number of paths is given by

$$P_n(x) = \begin{cases} P_{n-1} \left\lfloor \frac{x}{2} \text{ modulo } N \right\rfloor, & x \text{ even} \\ P_{n-1} \left\lfloor x - \frac{1}{2} \text{ modulo } N \right\rfloor + P_{n-1} \left\lfloor x + \frac{1}{2} \text{ modulo } N \right\rfloor, & x \text{ odd} \end{cases}$$

where $x = (D - S) \text{ modulo } N$, $P_1(0) = 1$, and $P_1(1) = 2$. Note that $P_n(0) = 1$ for all n and that $P_n(x) > 1$ for $x \neq 0$. Table 5.1 lists the number of paths as a function of the value of $(D - S) \text{ modulo } N$ [PaR82] for N up to 16.

The Gamma network can be controlled by n digit routing tags, the value of which is the difference modulo N between the numbers of the network input and output to be connected. The digits of the tag may be 1, 0, or -1, corresponding to the $+2^i$, straight, and -2^i links, respectively. Control of the Gamma network when faults occur is not explicitly specified.

The Gamma network can perform all permutations passable by the IADM network and some that it cannot. For example, the IADM cannot perform all permutations for $N = 8$ or the perfect shuffle for all $N \geq 8$ [SSM80, AdS82b], but the Gamma network can perform these permutations.

A fault model that can be used for the Gamma network assumes the following.

1. Only switching elements fail.

Table 5.1 Number of paths in the Gamma network for possible values of $(D-S)$ modulo N [PaR82].

<u>N</u>	<u>2</u>	<u>4</u>	<u>8</u>	<u>16</u>
<u>(D-S) modulo N</u>	<u>Number of Paths</u>			
0	1	1	1	1
1	2	3	4	5
2		2	3	4
3		3	5	7
4			2	3
5			5	8
6			3	5
7			4	7
8				2
9				7
10				5
11				8
12				3
13				7
14				4
15				5
<hr/>	<hr/>	<hr/>	<hr/>	<hr/>
Total	3	9	27	81

2. The input and output stage switching elements are always fault-free.
3. Faults occur independently.
4. Faulty switching elements are unusable.

The fault-tolerance criterion appropriate for the Gamma network is full access with the exception that an input need not be able to connect to the identically numbered output. Under this fault-tolerance model the network is single fault tolerant. For a computer system with a PE-to-PE structure, not requiring the ability to connect an input to an identically numbered output (an identity connection) is reasonable since a PE should not need to communicate with itself. For the P-to-M model on the other hand, identity connections are likely to be important.

5.2.7 Fault-Tolerant Beneš Network

The Fault-Tolerant Beneš Network is derived from the Beneš network [Ben65]. A Beneš network connects $N = 2^n$ inputs to N outputs via $2n - 1$ stages each with $N/2$ 2-input/2-output switching elements and is a particular instance of the more general Clos network [Clo53]. The switching elements can be set to one of two states: straight or exchange. Figure 5.11 shows the Beneš network for $N = 8$. The Beneš network is a *rearrangeable* network in that any idle input/output pair can be connected by rerouting any established one-to-one connections as necessary. In other words, any one-to-one connection can be established regardless of any existing one-to-one connections. Thus, the Beneš network can perform any permutation of inputs to outputs.

The fault model used in [SoR80] for the analysis of fault tolerance of the Beneš network is a switching element stuck-fault model. That is, a switching

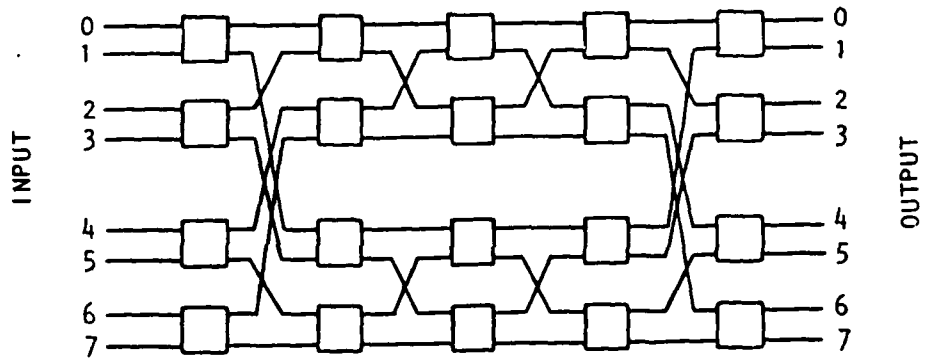


Figure 5.11 Beneš network for $N=8$.

element can be stuck in the straight setting, or stuck in the exchange setting. Specifically,

1. Only switching elements fail, and they fail by becoming stuck in one of their two states.
2. Faults occur independently.
3. Faulty switching elements are usable.

This is a relatively weak fault model, in that it supposes an optimistic view of hardware behavior. For example, other switching element failure modes may well be possible, such as ones where continued use of the switching element is not possible. Link failures may also occur in a physical network.

The Beneš network fault-tolerance criterion is retaining the ability to perform any permutation connection in a single pass through the network, known as *full connection capability*. It is the most stringent fault-tolerance criterion of all the networks surveyed, but the Beneš network is the most capable of these networks, in terms of permuting capability. The Beneš network can tolerate most single faults, as defined by this fault-tolerance model. Some multiple switching element faults not in the center stage can be tolerated as well, so the network is robust. However, if any single switching element in the center stage is stuck at the exchange setting then the identity permutation, which connects each input to the identically numbered output, cannot be performed. Also, if any center stage switching element is stuck at the straight setting then the uniform shift connecting each input i , $0 \leq i < N$, to output $i + N/2$ modulo N is one permutation no longer possible.

Any center stage fault is correctable by adding a single switching element at the input or output stage [SoR80]. The configuration of the fault-tolerant network with the extra switching element at the output is shown in Figure 5.12 for $N=8$. Tolerance of a fault is achieved by using the extra switching element to correct for the misrouting (if any) caused by the fault. Further modifications of the Beneš network allowing multiple-fault tolerance to switching element stuck-at faults, but requiring extra stages of switches, are described in [SoR80].

Fault-Tolerant Beneš network routing is performed by computing the necessary settings of all the switching elements, and then imposing that state on the network through control lines, one per switch. The algorithm for the Beneš network to compute control information for any permutation requires $O(N \log N)$ time for execution [OpT71] and global knowledge of the control state of the network, i.e., centralized control. The algorithm does not avoid faulty switches; required switch settings can be adjusted to match the state of a stuck switch. Faulty switches must be used if permutations are to be performed in only one pass through the network.

A new algorithm for controlling the Beneš network was presented in [Lee84]. Unlike the algorithm in [OpT71], it is not recursive and requires only knowledge of the control state of the previous stage, rather than the entire network. However, it too has $O(N \log N)$ time complexity, and also implies centralized network control.

In the Fault-Tolerant Beneš network, routing in the case of a fault not in a center stage switching element is performed as in the fault-free network, except that the reducible connection set assignment, which determines subnetwork settings, is changed, if needed, so that the subnetwork with the

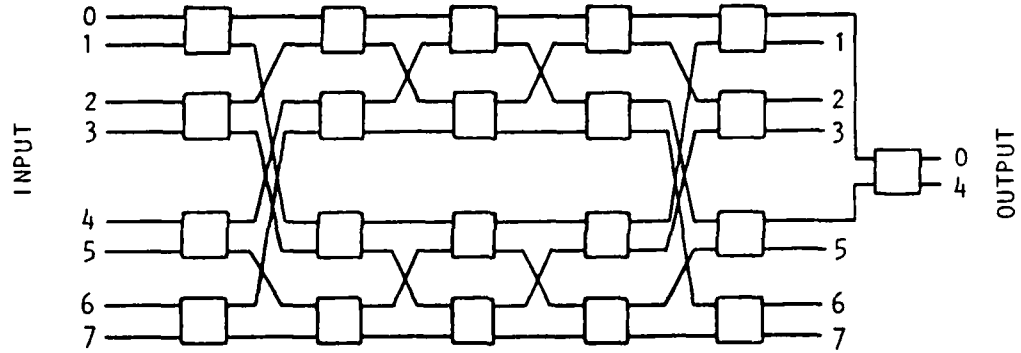


Figure 5.12 Fault-tolerant Beneš network for $N=8$.

stuck switch is assigned the reducible connection set that requires the faulty switch to be set as it is stuck [SoR80]. For permutations affected by a fault in the center stage, routing is accomplished by first computing the control signals that would be used in the Beneš network if outputs 0 and 4 of the permutation were interchanged. The outputs 0 and 4 are then exchanged by the additional switching element.

In general, the combination of centralized control and $O(N \log N)$ time make both the Beneš and the Fault-Tolerant Beneš networks unsuitable for a parallel/distributed system in which the network is often reconfigured.

5.2.8 Augmented C-Network

The Augmented C-network (ACN) is derived from the C-network [ReK84]. The C-network connects N inputs to N outputs via m stages, $m > 0$, of 2×2 switching elements, each stage with $N/2$ switches. Stages are numbered 0 to $m-1$ from input to output. C-networks are a broad family of interconnection networks. The Baseline, Omega, Generalized Cube, and Beneš networks are all instances of C-networks.

Let one output of a switch be labeled 0 and the other, 1. For a switch S in stage i , $i \neq m-1$, its *0-successor*, denoted $\text{succ}^0(S)$, is the switch in stage $i+1$ connected to its 0 output. The *1-successor*, $\text{succ}^1(S)$, is the stage $i+1$ switch connected to its 1 output.

The topology of a C-network is defined by the following relationship. For each switch S_j in stage i , $0 \leq i < m-1$, $0 \leq j < N/2$, there exists a switch S_k , $0 \leq k < N/2$ and $k \neq j$, in stage i such that $\text{succ}^0(S_j) = \text{succ}^0(S_k)$ and $\text{succ}^1(S_j) = \text{succ}^1(S_k)$. Switches S_j and S_k are said to be *conjugate*; this is

denoted as $\text{conj}(S_j) = S_k$ and $\text{conj}(S_k) = S_j$.

The notion of conjugate switches was discussed in [Agr83] using the term *output buddies*. The concept of *input buddies*, the 0- and 1-successors of a switch and its conjugate, was also noted. A network in which two pairs of input buddies also constitute two pairs of output buddies has the *strict buddy* property. C-networks do not necessarily have this property.

A C-network with the additional property that $\text{succ}^0(S) \neq \text{conj}(\text{succ}^1(S))$ and $\text{succ}^1(S) \neq \text{conj}(\text{succ}^0(S))$ is the basis for the ACN. So that all networks in this survey can be compared on an equal basis the C-network will be assumed to have $n = \log_2 N$ stages. Beginning with such a C-network, an ACN is constructed by replacing all the 2×2 switching elements with 4×4 crossbar switches with inputs labeled 0, 1, $\text{conj}(0)$, and $\text{conj}(1)$. Switch outputs are labeled similarly (see Figure 5.13). Those switch inputs and outputs labeled 0 and 1 are connected exactly as in the base C-network. The $\text{conj}(0)$ and $\text{conj}(1)$ ports are connected as follows. In stage 0, switch S inputs $\text{conj}(0)$ and $\text{conj}(1)$ are connected to the sources connected to the 0 and 1 inputs of switch $\text{conj}(S)$, respectively. This is shown in Figure 5.14(a). In stage $n-1$, switch S outputs $\text{conj}(0)$ and $\text{conj}(1)$ are connected to the outputs connected to the 0- and 1-outputs of switch $\text{conj}(S)$, respectively (see Figure 5.14(b)). The $\text{conj}(0)$ and $\text{conj}(1)$ outputs of switches S_a and $\text{conj}(S_a)$ in stage i , $0 \leq i < n-1$, are connected to the $\text{conj}(0)$ and $\text{conj}(1)$ inputs, respectively, of $\text{conj}(\text{succ}^0(S_a))$ and $\text{conj}(\text{succ}^1(S_a))$, where $\text{succ}^0(S_a) = S_b$ and $\text{succ}^1(S_a) = S_c$ (see Figure 5.14(c)).

The ACN fault model implied in [ReK84] is the following.

1. Both switching elements and links fail.

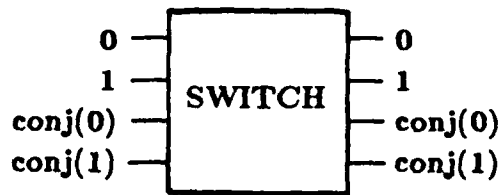


Figure 5.13 Switching element of an ACN network.

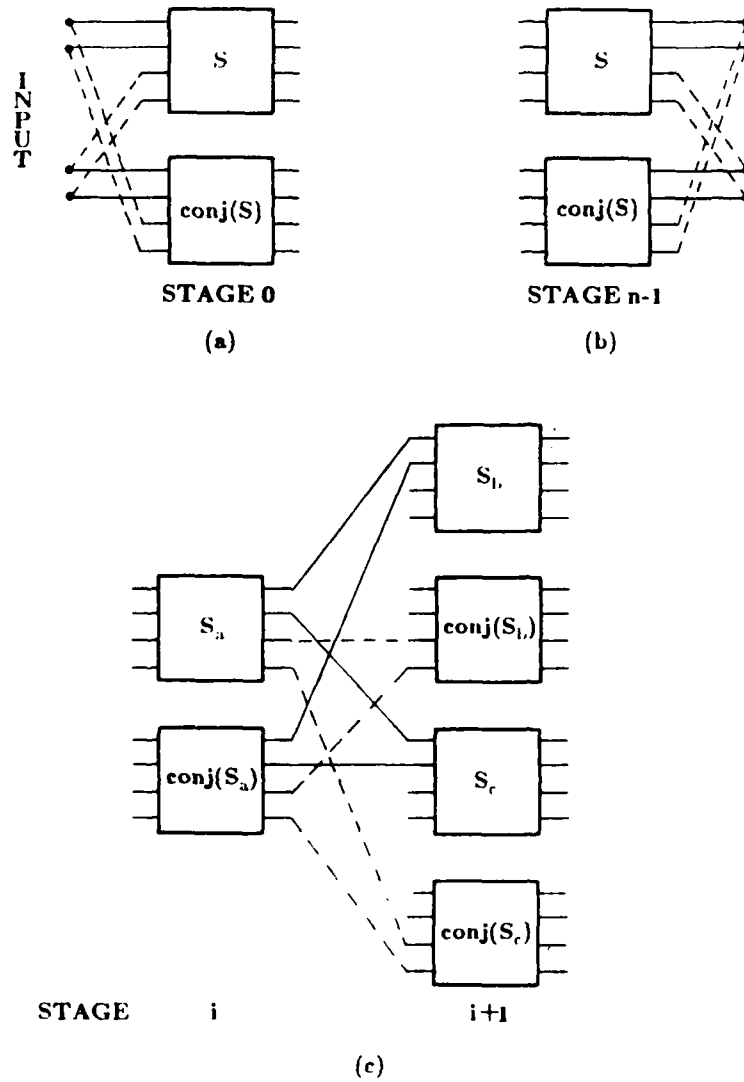


Figure 5.14 (a) Connections to stage 0 switches in an ACN. (b) Connections from stage $n-1$ in an ACN. (c) Connections from a conjugate switch pair in an ACN.

2. Faulty switching elements are usable.

The ACN fault-tolerance criterion is retention of full access capability. The ACN provides 2^n distinct paths between any source and destination, however, most of these paths are not disjoint. In general, there are at least two switches at any stage and two links between stages by which a given source and destination can be joined. Thus, the ACN is single-fault tolerant to both switch and link failures.

Routing in the ACN is predicated on a routing tag scheme existing for the base C-network. It is assumed that a switch can determine when a successor switch is faulty. In the case of no faults, the C-network routing tag is determined and interpreted as for the base C-network; the *standard path* is taken through the ACN.

In case there are unavailable switches (either due to failure or previously established data transfer) two routing strategies are proposed [ReK84]. Each strategy begins with the routing tag for the base C-network. The first strategy is defined by the flow chart shown in Figure 5.15. The second strategy is similar to the first, and is shown in Figure 5.16. Either routing scheme takes full advantage of the fault tolerance characteristics of the ACN.

5.2.9 β -Networks

A β -network is formed by interconnecting a set of β -elements [ShH84]. A β -element is a 2-input/2-output switch that can perform the two permutations straight and exchange. That is, a β -element is a two-state interchange box [SiS78, Sie79a]. Thus, many networks, including the Fault-Tolerant Beneš and Modified Baseline, are β -networks. An example of a simple β -network is shown

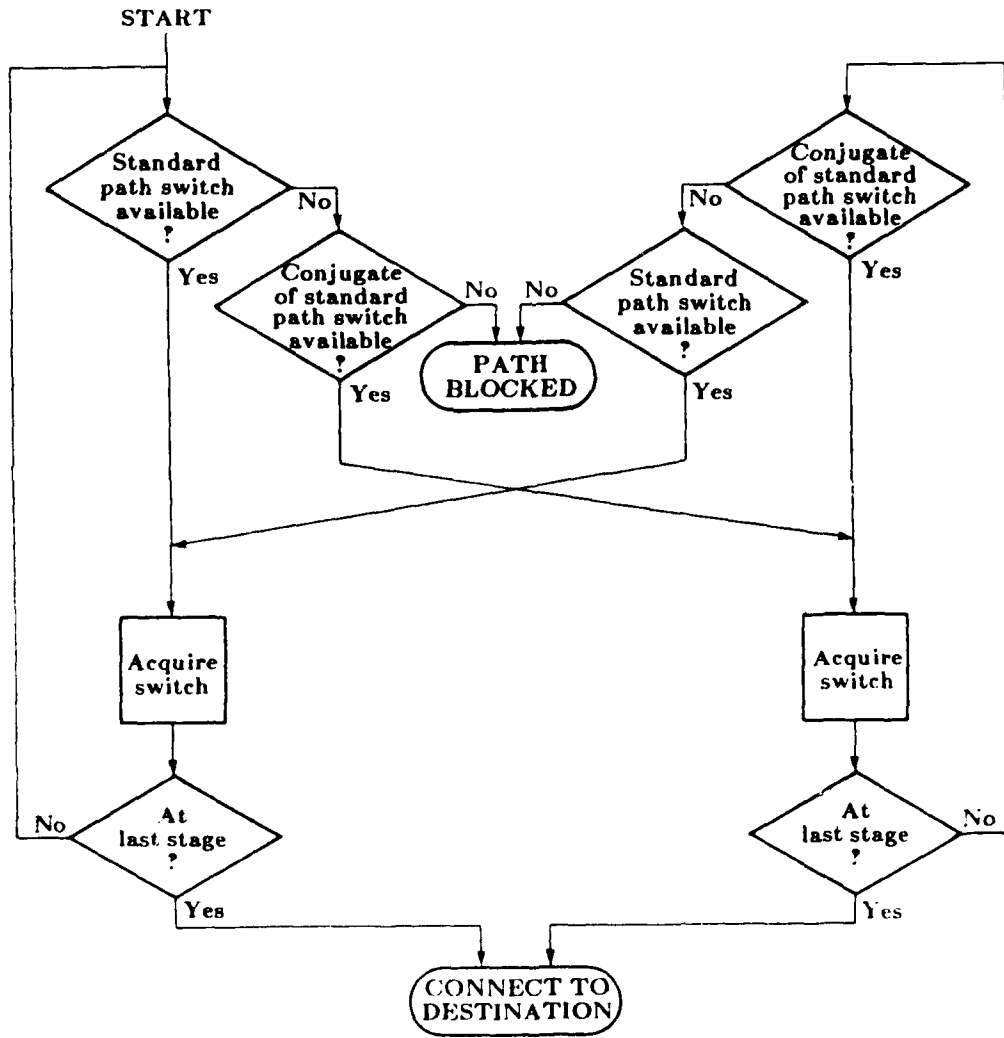


Figure 5.15 Flow chart for one ACN routing strategy.

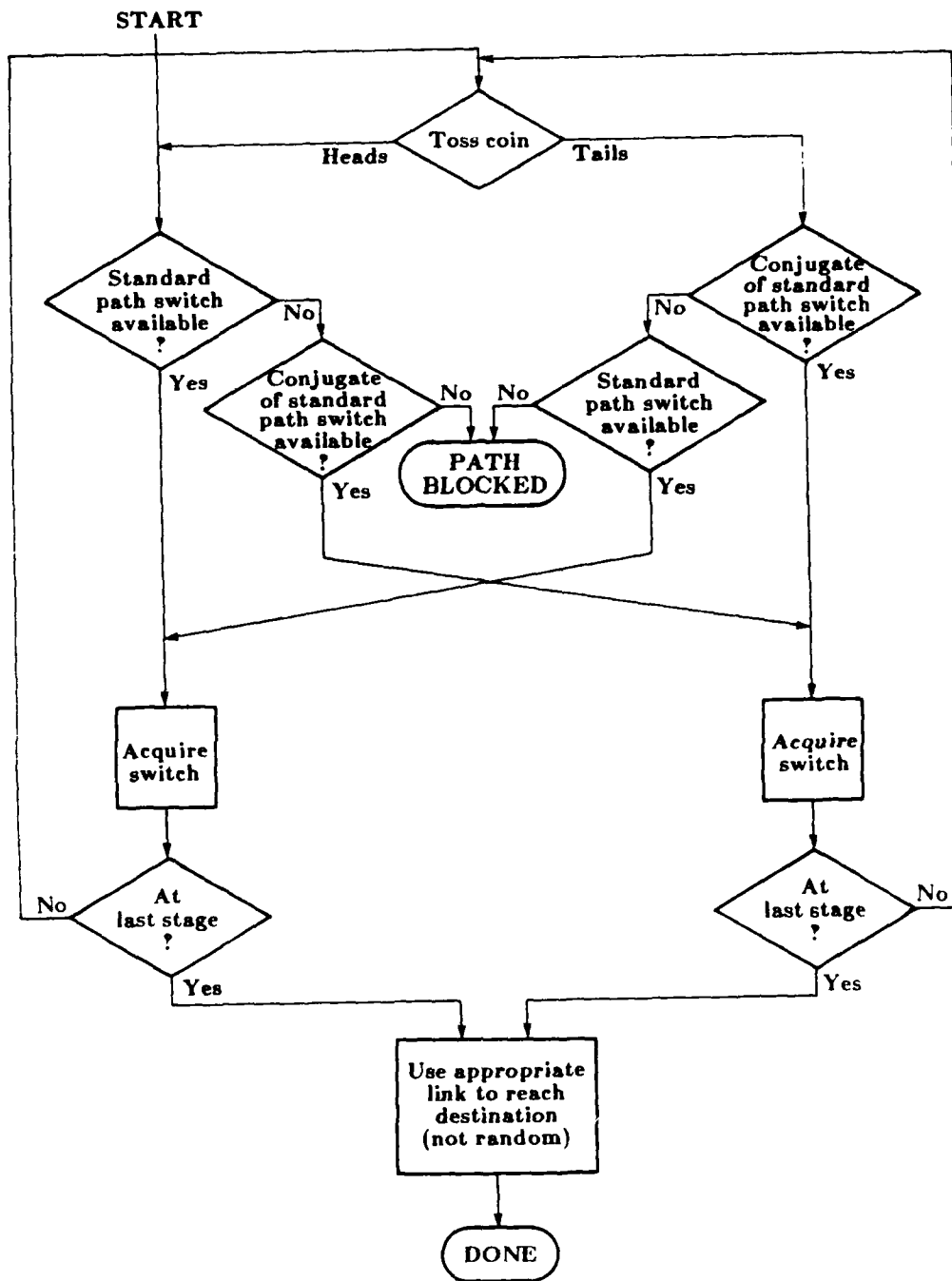


Figure 5.16 Flow chart for another ACN routing strategy.

in Figure 5.17. Although they may duplicate the topologies of other surveyed networks, β -networks are included for their fault model and fault-tolerance criterion, as these differ substantially from those of the other networks considered in this chapter. This provides a more complete view of the state of the art in fault-tolerant multistage networks. In the sense that β -networks can be designed to enhance their ability to tolerate faults, they are included in this survey of networks so designed.

A β -network is defined as having the *dynamic full-access* property if any given network input can be connected to any single network output in a finite number of passes through the network. Between passes it is assumed that each output can connect to its corresponding input (i.e., the input with the same number as the output) via a path outside the network. The β -network is said to tolerate a fault if the fault does not destroy dynamic full-access capability. This is a considerably less restrictive fault-tolerance criterion than those of the other networks surveyed. The purpose in using the dynamic full-access measure is to better characterize the connectivity requirements of computer systems than either full-access or rearrangeability (full connection) capability [ShH80]. However, the multiple pass method of network operation implied by the dynamic full access criterion may be unsuited for some, if not many, applications.

The fault model used for β -networks includes two failure modes in the β -element only. These failures are stuck at straight and stuck at exchange. The β -elements are assumed to remain able to pass data despite faults. This is the same failure mode assumed for switching elements in the Fault-Tolerant Beneš network.

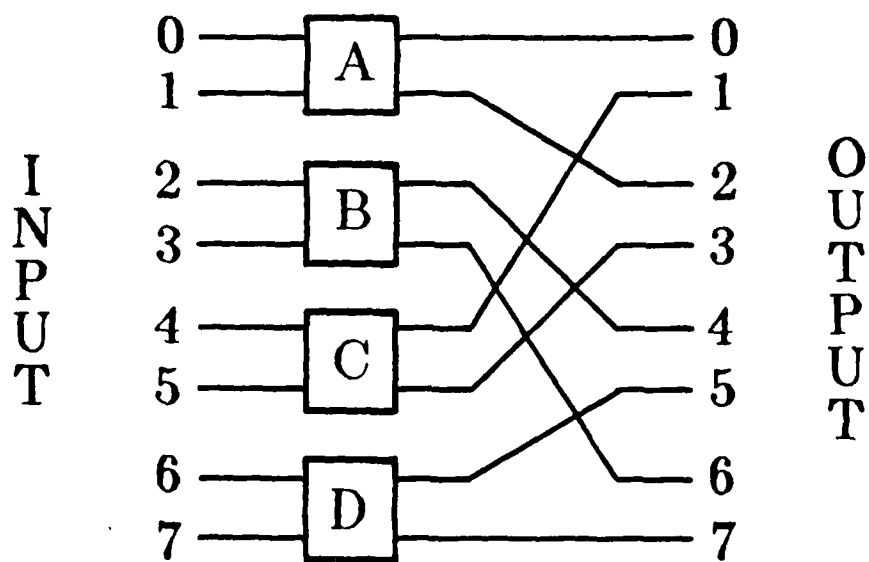


Figure 5.17 A simple single-stage β -network.

A graph model is used to study faults and fault tolerance in β -networks. A β -graph is a directed graph with vertices representing the β -elements, and edges representing the links in the β -network. There is an edge from vertex i to vertex j of the β -graph if an output terminal of β -element i connects to an input terminal of β -element j . Figure 5.18 shows the β -graph of the example β -network in Figure 5.17. For multistage β -networks, edges in the β -graph can denote network links or devices using the network. The edges representing devices using the network are called *computer edges*.

A *critical fault* is a collection of stuck β -elements which destroys the dynamic full-access property of the β -network. A faulty β -element is stuck at either straight or exchange, so in the β -graph it can be represented as a split vertex. A *strongly connected* directed graph is one such that between each pair of vertices a and b there exists a path from a to b as well as from b to a [JoJ72]. Thus, the dynamic full-access property is lost by a fault which disconnects the originally strongly connected β -graph. A network design that retains dynamic full access for the maximum number of faults is discussed in [AgL84].

A *minimal critical fault* is a critical fault for which no proper subset constitutes a critical fault. Minimal critical faults can be characterized in two ways. One is by *circuit partitions*, a collection of edge-disjoint elementary circuits of a β -graph such that each vertex belongs to exactly two elementary circuits and each edge belongs to exactly one elementary circuit. Figure 5.19 illustrates two circuit partitions of the example β -network.

For every circuit partition there is a circuit *adjacency graph* which is an undirected graph whose vertices represent the elementary circuits of the partition, and whose edges represent the vertices of the β -graph. Figure 5.20

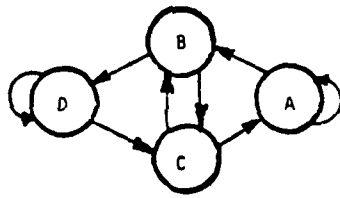


Figure 5.18 The β -graph of the example single-stage β -network [ShH80].

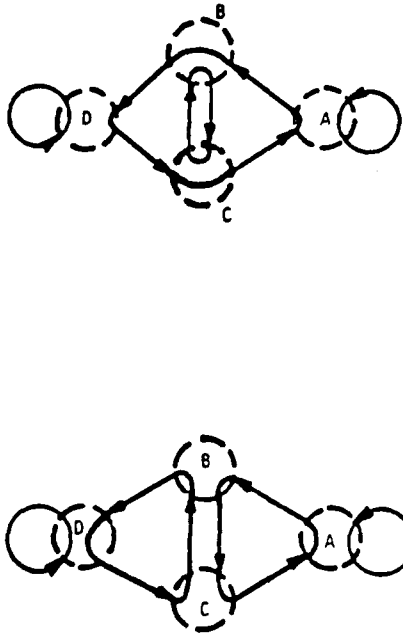


Figure 5.19 Two circuit partitions of the example β -network [ShH80].

shows the two circuit adjacency graphs for the example β -network circuit partitions.

A *cutset* is a minimal set of edges in a graph with p separate parts, the removal of which results in a graph with $p + 1$ parts [JoJ72]. Every cutset of a circuit adjacency graph represents a minimal critical fault. Also, every minimal critical fault can be represented by a cutset of a circuit adjacency graph. That is, a one-to-one correspondence between circuit adjacency graphs and minimal critical faults exists.

A second way to characterize minimal critical faults is with Eulerian circuits, which are circuits that use every edge in a directed graph once [JoJ72]. Figure 5.21 gives the two Eulerian circuits of the example β -graph. A β -network fault is critical if it is not compatible with any Eulerian circuit of the β -network. Two network settings are *compatible* if all specified β -element settings (straight or exchange) match. Unspecified β -element settings ("don't care" settings) always match. A fault can be considered a partial setting where the faulty β -elements are specified and the remainder are "don't care."

There are two important disadvantages to the β -network approach to fault-tolerant networks. One is the computational complexity of using the critical fault characterizations. Even when faults have been detected and located considerable work remains to determine the operational status of the network. Specifically, the set of located faults must be tested to see if it comprises a critical fault. This can be done by checking to see if any subset of the faults constitutes a minimal critical fault. A specific testing technique is presented in [AgL84]. A list of minimal critical faults can be precomputed for a given β -network structure. The second disadvantage is that by allowing a finite number of passes through the network, data transit time becomes

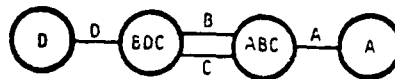
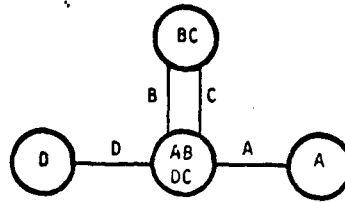


Figure 5.20 The two circuit adjacency graphs for the two circuit partitions, respectively [ShH80].

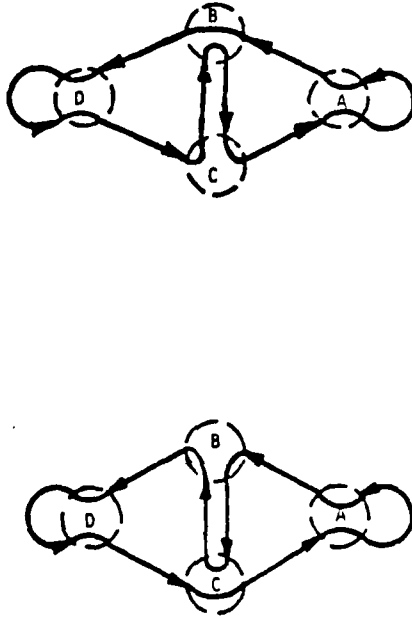


Figure 5.21 Two Eulerian circuits of the β -graph of the example β -network [ShH80].

variable. This will impose burdens on an SIMD system attempting to maintain synchronization.

Routing in a β -network can be accomplished using binary routing tags with as many bit positions as there are stages in the network. However, β -networks constitute such a broad class that no one routing tag scheme is generally applicable. Also, realization of dynamic full access capability may incur significant computational expense for routing tags, since a set of tags leading from the original source via a finite number of passes through the network to the ultimate destination must be generated. No scheme for generating such a set of tags is given in [ShH80].

5.3 Comparison

Table 5.2 summarizes the network fault tolerance information presented in Section 5.2. It gives the possible faults that can occur in each network under the assumed fault model, whether or not faulty components are usable, the fault-tolerance criterion, the method by which the network copes with faults, whether the network is single-fault tolerant, and how it performs when there are multiple faults. Note that the phrase "internal node faults only" used in the table is another way of saying input and output switching elements are always fault-free.

There is a growing literature on fault-tolerant multistage interconnection networks as is demonstrated by the survey of Section 5.2. However, as pointed out in [LLY82] the results have several limitations, including:

1. unreasonably optimistic fault-tolerance models, and

Table 5.2 Summary of fault tolerance information for the networks surveyed.
 "SE" is an abbreviation for switching element.

Network	Fault Model	Fault-Tolerance Criterion	Fault-Tolerance Method	Single-Fault Tolerant [†]	Multiple-Fault Tolerance [†]
Modified Baseline	internal SE only; unusable	full access	alternate route	yes	robust
Augmented Delta	internal SE or link; unusable	full access	alternate route	yes	robust; (b-1)-fault tolerant with b x b switches
Multipath Omega	internal SE or link; unusable	full access	alternate route	yes	robust
F-network	internal SE only; unusable	full access	alternate route	yes	robust
Enhanced IADM (straight links)	internal SE or link; unusable	full access	alternate route	no	robust
Enhanced IADM (half links)	internal SE or link; unusable	full access	alternate route	yes	robust; 2-fault tolerant with look ahead
Gamma	internal SE only; unusable	full access, but no identity connection	alternate route	yes	robust
Fault-Tolerant Beneš	SE stuck, but usable	full connection capability	correct misroute	yes	robust
Augmented C-network	any SE or link; unusable	full access	alternate route	yes	robust
β -networks	SE stuck, but usable	dynamic full access	repeated pass	depends on network	typically robust

[†] Answer depends critically on fault model and fault-tolerance criterion.

2. increased data routing complexity.

To place the ESC in perspective, it is compared with the fault-tolerant multistage interconnection networks surveyed in Section 5.2. Table 5.3 summarizes that comparison. Note that Table 5.3 gives qualitative information; for example, in the column on fault model the phrase "slightly less strict" generally means input and output stage switching elements are assumed fault-free, while the phrase "less strict" typically includes the previous restriction and adds the assumption of fault-free links. For specific details see the appropriate subsection of Section 5.2. The facts and reasoning supporting Table 5.3 are discussed below.

As noted in Chapter 4, the choice of fault model and fault-tolerance criterion plays a key role in determining the fault-tolerance characteristics of a network. ESC fault tolerance is evaluated in light of a fault model that presupposes the possibility of failure of any network component except the stage n demultiplexers and stage 0 multiplexers. (Stage n multiplexers and stage 0 demultiplexers are treated as stage n and stage 1 link faults, respectively.) As can be seen from Table 5.3, this fault model is stricter than all but one of the fault models of the comparison networks. The ESC fault model assumes at least as many possibilities for failure within a network (both switching elements and links) and dire consequences for such failures (any faulty component is unusable). It may well be the most realistic of those fault models.

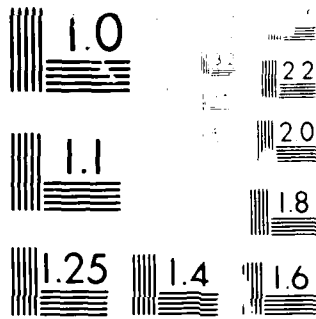
The fault-tolerance criterion for the ESC is the same as that for most of the networks surveyed. Basically, what is required is that one-to-one interconnection capability be uncompromised.

Table 5.3 Comparison of surveyed networks with the ESC. Entries give the relationship between the network in question and the ESC as regards a particular attribute. "SE" is an abbreviation for switching element.

Network	Fault Model	Fault-Tolerance Criterion	Routing Complexity	Hardware Complexity	Fault-Tolerance Capability*
Modified Baseline	less strict	same	similar	slightly less	similar
Augmented Delta	slightly less strict	same	similar	slightly less	similar
Multipath Omega	slightly less strict	same	similar to slightly greater	slightly less	similar
F-network	less strict	same	similar	slightly greater	similar
Enhanced IADM (straight links)	slightly less strict	same	similar	greater	less
Enhanced IADM (half links)	slightly less strict	same	similar; complexity hidden in SE	greater	greater if complex routing used
Gamma	less strict	same	similar	greater	similar
Fault-Tolerant Beneš	much less strict	stricter	much higher	much greater	similar
Augmented C-network	as strict	same	similar; complexity hidden in SE	much greater	similar
β -networks	much less strict	much less strict	much higher	less to greater [†]	similar to greater [†]

* Using the fault model and fault-tolerance criterion defined for that network.

† Depends on specific instance of network structure.



The fault-tolerance criterion of the Gamma network may be unsuitable for some computer systems as it does not consider inability to connect an input to the identically numbered output (an identity connection) to be a failure. If the same device, e.g., a PE, is connected to the same numbered input and output then this is not a problem, since a device should not need to communicate with itself. However, a parallel processor with a P-to-M structure does require a network that can connect a processor to any memory module. With the Gamma network, prudence would dictate that a processor and any memory module it will access particularly often not be placed on network ports with the same numbers (access to the memory module with the same port number would be through an intermediate memory module and processor).

The Fault-Tolerant Beneš network uses the more demanding criterion that *permuting capability be unaffected*: all permutations should still be performable with a single pass through the network. It is reasonable to use this strict criterion because the Fault-Tolerant Beneš network is the most capable interconnection network surveyed in the sense that it is the only one with full connection capability.

The fault-tolerance criterion used to study β networks is a much less strict test to pass. All that is required is that it be possible to connect any input to any output in a finite number of passes through the network. Successive passes are performed by returning data from a network output to the same numbered input. Even a fault-free β network may require multiple passes for data to reach its destination, so the chosen fault-tolerance criterion is appropriate. However, since the class of β networks is so broad, it is important to note that this forgiving criterion may inflate the capabilities attributed to more complex β networks.

For most of the networks, routing in the presence of faults is little more complex than in the absence of faults. The notable exception to this is β -networks. Given an input and output to be connected, the dynamic full access procedure requires choosing a set of intermediate outputs which can each be reached consecutively. A general solution to this problem is not known. Routing complexity for the Fault-Tolerant Beneš network is higher than for the ESC because of the nature of the Beneš network [OpT71]; it is not due to the modification for fault tolerance.

Hardware complexity of the networks varies significantly. The measure underlying the presentation in Table 5.3 is a mixture of the asymptotic number of network components needed and the complexity of the switching element. The intent is to get a rough estimate of package count at various levels, specifically, the chip and board level or multiple-chip carrier level (e.g., the IBM Thermal Conduction Module [BIB82]). The obvious use of such information is to assess implementation cost. However, hardware complexity and implementation cost may bear little relationship; knowledge of implementation details arising from a hardware design study is necessary for high-confidence estimates of cost. VLSI technology often allows a large reduction in package count, for example. With this caveat, the hardware complexity data can be used for comparison purposes.

The fault-tolerance capabilities of the networks are all reasonably similar given the chosen standard by which each network's features are determined. This is apparent in the column on fault-tolerance capabilities in Table 5.3. There should be no surprise that this is so. It is easy to agree with the idea that it is desirable for a network to have whatever fault-tolerance capabilities are feasible. Single-fault tolerance is more feasible than i -fault tolerance, $i > 1$.

However, because each network is studied using its own fault model and fault-tolerance criterion, significant differences in capabilities might appear if a common fault model and fault-tolerance criterion are adopted.

The ESC fault model and fault-tolerance criterion can be applied to the networks surveyed in the previous section in order to relate their fault tolerance to that of the ESC network. This information is given in the first column of Table 5.4. Under the ESC fault model and fault-tolerance criterion only the ESC and ACN are single-fault tolerant. Many of the networks fail to be single-fault tolerant because they cannot tolerate an input or output switching element fault, as can the ESC and ACN. This is why so many of the fault models refer only to internal switching element faults. If the ESC fault model is amended to assume fault-free switching elements in the input and output stages, some of the networks become single-fault tolerant as shown in the table. Alternatively, these same networks that are single-fault tolerant under the relaxed fault model could be fitted with input and output stage bypass circuitry and become single-fault tolerant without weakening the fault model.

The Erlang or FADM with additional straight links is not single-fault tolerant when the ESC fault model is relaxed because it still cannot tolerate all switching element failures. This includes the switching element failures in internal stages, all will fail for the Erlang link model. The additional straight links provide fault tolerance against failures of a straight link, but not a switch. The fault-tolerance capability with respect to switches is the same as the FADM, and there are many cases where a switch failure will block a connection [MSS2a].

Table 5.4 Fault tolerance capabilities of the networks using the ESC network fault model and fault tolerance criterion.

- A. Single-fault tolerant using ESC fault model and fault-tolerance criterion.
- B. Single-fault tolerant if ESC fault model is relaxed to assume input and output stage switching elements are fault-free.

Network	A	B
Extra Stage Cube	yes	yes
Modified Baseline	no	yes
Augmented Delta	no	yes
Multipath Omega	no	yes [*]
F-network	no	yes
Enhanced IADM (straight links)	no	no
Enhanced IADM (half links)	no	yes
Gamma	no	no
Fault-Tolerant Beneš	no	yes [†]
Augmented C-network	yes	yes
β -networks	no	yes [*]

^{*} Typically yes, but depends on specific instance of network structure.

[†] Must modify control scheme to achieve fault tolerance under this model.

The Gamma network is not single-fault tolerant under the relaxed model because it has only one path from an input to the identically numbered output. The ESC fault-tolerance criterion is full access, so a straight link fault will prevent an input from communicating with the identically numbered output (as it would in the IADM network on which the Gamma network is based). Thus, the Gamma network does not satisfy the ESC fault-tolerance criterion of full access.

The Fault-Tolerant Beneš network is capable of single-fault tolerant operation under the relaxed ESC fault model. Although faulty components cannot be used to pass data under the ESC fault model (unlike the Fault-Tolerant Beneš fault model), only one-to-one connections need be supported (as compared to permutation connections for the Fault-Tolerant Beneš fault model). The Fault-Tolerant Beneš network can perform any one-to-one connection without using a given faulty component. However, the control method given in [SoR80] must be modified to achieve this fault tolerance capability so that faulty network components are avoided (the given algorithm uses faulty components).

The ACN is single-fault tolerant under the ESC fault model and fault-tolerance criterion. However, this capability comes at the price of significantly increased hardware complexity. ACN 4×4 crossbar switching elements are more complex than the interchange boxes used in the ESC and must have redundancy in addition to that needed to support the routing scheme. Further, the ACN has twice as many links as the ESC. For even small network sizes, the ESC is likely to possess an implementation cost advantage over the ACN.

A 3 network is single fault tolerant given the relaxed ESC fault model if it provides at least two paths that share no network components other than

their first and last β -elements between any source and any destination. This restriction must hold for all passes through the network that may be required for the two paths.

5.4 Conclusions

Nine fault-tolerant multistage interconnection networks have been described. The particular characteristics considered included fault model, fault-tolerance criterion, fault-tolerance method, single- and multiple-fault tolerance, routing complexity, and hardware complexity. With this background, the networks were compared with the ESC.

There are wide variations in topology and switching element design among the networks surveyed. The Augmented Delta, Modified Baseline, and Multipath Omega networks are cube-type networks. The Augmented Delta and Modified Baseline networks each incorporate an extra stage of switching elements to provide redundant paths. The Multipath Omega may use an extra stage or stages of switching elements, or substitute a stage or stages of different switches, or use some combination of these methods. The Gamma network uses 3×3 crossbar switching elements and the same link connection pattern as the IADM. The Enhanced IADM and F-network use 5×5 and 4×4 switches, respectively, which pass one item at a time. The Augmented C-network uses 4×4 crossbar switching elements with a general family of topologies. The Fault-Tolerant Beneš network and β -networks are both composed of β -elements, but have different link patterns.

The ESC network provides single-fault tolerance despite its challenging fault model and fault-tolerance criterion. The fault model and fault-tolerance criterion were chosen for their consistency with engineering pragmatism. It is

straightforward to implement the fault-tolerance capabilities of the ESC, and its hardware complexity is competitive with other fault-tolerant, multistage networks. The ESC is intended to be a viable answer to real parallel computer interconnection needs.

CHAPTER 6

RELIABILITY OF THE EXTRA STAGE CUBE INTERCONNECTION NETWORK

6.1 Introduction

The term *reliability* has many different technical meanings; one that is commonly accepted is that the reliability of a component or system at time t , denoted $R(t)$, is given by $R(t) = P\{T > t\}$, where T is the time to failure and $P\{T > t\}$ is the probability that $T > t$ [Mey70]. Time to failure can be considered a continuous random variable of some probability density function. The failure behavior of components can be studied and a probability density function chosen to represent that behavior as accurately as possible.

An analogous reliability measure for interconnection networks, termed *terminal-pair reliability* [RaP84] has been applied to the Gamma network [PaR82, PaR84]. Terminal-pair reliability is the probability that there exists at least one path between any network input and output. This reliability measure is appropriate for the ESC as well as the Gamma network. For the ESC it can be calculated as one minus the probability of loss of fault-free interconnection capability. In the case of one fault, the ESC was shown in Chapter 4 to have a zero probability of such loss of capability, hence, a reliability of one. The goal here is to assess multiple-fault tolerance of the ESC [AdS84].

6.2 Fault Model

The fault model chosen for the ESC provides a strong challenge to fault tolerance analysis because it supposes both boxes and links fail, and fail with different likelihoods. The complex nature of a physical network is thus more accurately captured, but incorporating this complexity in an analysis is correspondingly more involved.

As with any fault model, the accuracy with which reliability predictions match measured reliability is the touchstone for the validity of the model. The assumptions made for the model used here are more stringent with respect to the analysis to be presented than those typically used for similar problems. The possibility of link failure is often ignored in the literature on fault-tolerant multistage interconnection networks [Agr82, Ci82, PaR82, ShH80, SoR80, WFL82], fault tolerance analysis [She82], and network reliability [Ci82].

6.3 Study of Multiple-Fault Tolerance

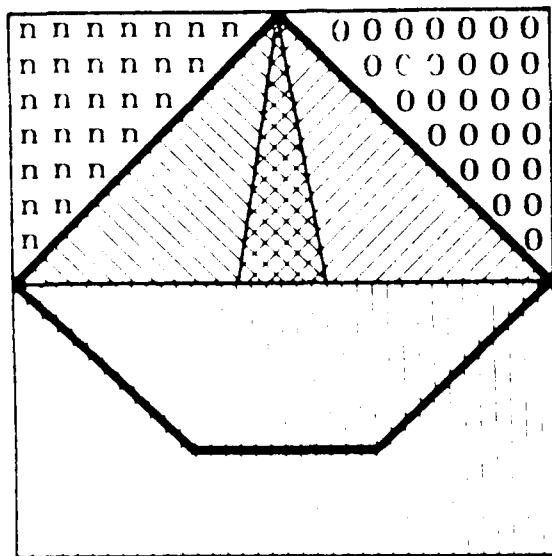
Because stage n and 0 interchange boxes are bypassed when faulty, an analysis of the probability of loss of fault-free interconnection capability of the ESC can be naturally structured in four cases based on classes of physical fault grouping. The four cases are closely related to the five fault categories listed in Table 4.1. The first case is derived jointly from the first and second categories of that table. The remaining three cases correspond to the third, fourth, and fifth categories, respectively.

Let f be the total number of box and/or link faults present in an ESC network. For the first case assume either all f faults are stage n box faults or all faults are stage 0 box faults. Fault-free interconnection capability is retained in this situation, although only one path remains between any source

and destination because the stage with all of the faults will be disabled. In the second case at least one, but not all f , faults are stage n box faults. Fault-free interconnection capability is lost in this circumstance because stage n is disabled, ensuring that the additional fault(s) must affect the single path between some source and destination. An analogous situation exists if at least one, but not all f , faults are stage 0 box faults. Finally, if all f faults are in network components other than stage n or 0 boxes, then Theorem 4.6 can be used to determine whether fault-free interconnection capability is available.

The union of the sets of pattern of faults associated with these four cases is the set of all possible occurrences of faults in the ESC. Figure 6.1 illustrates the relationships among the cases. The set of all possible outcomes of an experiment is the *event space*, of that experiment [Mo70], hence Figure 6.1 illustrates the event space of fault occurrences for the ESC. The first case is represented by the symbols "n" and "0." The second and third cases intersect, representing situations with b_n stage n box faults and b_0 stage 0 box faults, $1 \leq b_n, b_0 \leq f$ and $b_n + b_0 \leq f$. These cases are denoted by the two directions of diagonal shading. The last case is indicated by the vertical shading. Loss of fault-free interconnection capability is associated with those physical fault groupings that fall within the region of the diagram enclosed by the bold line. Note that the area devoted to a case is not intended to be proportional to the number of fault groupings fitting the description of that case, but rather to allow easy viewing of the relationships between the cases.

The probability that f faults cause a loss of fault-free interconnection capability is a function of N , as well as f . For example, Figure 4.4 shows an ESC network for $N = 8$ that retains fault-free interconnection capability despite the presence of $f = 2$ (labeled (2,2) and (1,1)). However, an ESC network for






- Case 1: $\left\{ \begin{array}{l} n \\ n \ n \end{array} \right.$ - All faults in stage n boxes
- $\left\{ \begin{array}{l} 0 \\ 0 \ 0 \end{array} \right.$ - All faults in stage 0 boxes
- Case 2:  - At least one but not all faults in stage n boxes
- Case 3:  - At least one but not all faults in stage 0 boxes
- Case 4:  - All faults in network components other than stage n or stage 0 boxes

Figure 6.1 Event space for classes of physical fault groupings in the ESC. The region characterized by loss of fault-free interconnection capability is enclosed by the bold line (drawing not to scale)

$N = 2$ will lose fault-free interconnection capability with any two faults. This is because there are only two boxes (one each for stages n and 0) and two links in this network. So, either both links are faulty, eliminating all paths through the network, or a stage n (or 0) box fault is paired with some other fault. The latter corresponds to the second (or third) fault grouping case.

The following notation is used in the subsequent equations. $P(X)$ denotes the probability that event X occurs. An *event* is a subset of the set of all possible outcomes of an experiment [Mey70]. If X is certain to occur, then $P(X) = 1$. $P(X) = 0$ implies X is not possible. The probability of X given an event A has happened is written $P(X|A)$. Let b be the total number of box faults, and b_i be the number of stage i box faults in a faulty ESC.

The probability of loss of fault-free interconnection capability is written $P(\text{loss} | N, f)$. In general,

$$P(\text{loss} | N, f) = P(\text{case 2} | N, f) + P(\text{case 3} | N, f) +$$

$$P(\text{case 2 and case 3} | N, f) + P(\text{case 4 and loss} | N, f)$$

More formally,

$$P(\text{loss} | N, f) = P(\{b_i < b_n, i = 1, \dots, n-1\} | N, f) + P(\{b_i < b_0, i = 1, \dots, n-1\} | N, f) +$$

$$P(\{b_i < 1, \text{ for } i = 1, \dots, n-1\} | N, f) +$$

$$P(\{b_i < b_n = 0 \text{ and loss}\} | N, f) \quad (6.1)$$

The first term in (6.1) is the probability of the event $b_i < b_n$ relative to the situation where b_n is the number of faults associated with box faults. The second term is the probability of the event $b_i < b_0$ relative to the situation where b_0 is the number of faults associated with link faults. The third and fourth terms

The last term covers all other ways fault-free interconnection capability can be lost.

It is useful to establish some basic relationships before proceeding with the evaluation of $P(\text{loss} \mid N, f)$.

$$P(\text{box fault} \mid \text{a fault}) = 1 - P(\text{link fault} \mid \text{a fault}),$$

since a fault must be in either a box or link. Note that strictly

$$P(\text{box fault} \mid N, \text{a fault}) \neq P(\text{box fault} \mid \text{a fault}),$$

because

$$\begin{aligned} P(\text{box fault} \mid N, \text{a fault}) &= \frac{\text{number of boxes}}{\text{number of boxes and links}} * P(\text{box fault} \mid \text{a fault}) \\ &= \frac{n+1}{3n+1}, \end{aligned}$$

a function of N ($n = \log N$). However, the ratio of the number of boxes to the number of boxes and links, $\frac{n+1}{3n+1}$, rapidly approaches the value $\frac{1}{3}$ for increasing N . Because of this behavior in the limit, and the assumption that all boxes have the same reliability regardless of other circumstances, the approximation

$$P(\text{box fault} \mid N, \text{a fault}) = P(\text{box fault} \mid \text{a fault})$$

is used to simplify the analysis. Similarly, the approximation

$$P(\text{link fault} \mid N, \text{a fault}) = P(\text{link fault} \mid \text{a fault})$$

is used. Note that if stage 0 were assumed to have links (which would connect the network to its output ports), then the two above approximations become

exact relationships. For notational compactness $P(\text{box fault} \mid \text{a fault})$ and $P(\text{link fault} \mid \text{a fault})$ will hereinafter be written $P(\text{BF} \mid \text{F})$ and $P(\text{LF} \mid \text{F})$, respectively.

Let A be an event and \bar{A} its complementary event. That is, \bar{A} denotes the set of all possible outcomes other than those in A . Assume an experiment is repeated k times with $P(A)$ constant for all repetitions. Define x to be the number of times A occurs. Then x is the *binomial random variable* with parameters k and $P(A)$, and

$$P(x = i) = \binom{k}{i} \cdot P^i(A) \cdot P^{k-i}(\bar{A}) = \binom{k}{i} \cdot P^i(A) \cdot (1 - P(A))^{k-i},$$

for $i = 0, 1, \dots, k$ [Mey70]. Recall $\binom{a}{b} = \frac{a!}{(a-b)!b!}$, for $a \geq b \geq 0$, where $0! = 1$. By definition $\binom{a}{b} = 0$ if $a < b$ or $a < 0$ or $b < 0$. The number of boxes in the network is $N(n+1)/2$, so $0 \leq b \leq N(n+1)/2$ (where b is the number of box faults); the number of boxes and links is $N(3n+1)/2$, so $0 \leq f \leq N(3n+1)/2$. Combining the relationship for the binomial random variable with the approximation $P(\text{box fault} \mid N, \text{a fault}) = P(\text{BF} \mid \text{F})$ yields the following.

$$\begin{aligned} P(b=j \mid N, f) &= P(b=j \mid f) \\ &= \binom{f}{j} \cdot P^j(\text{BF} \mid \text{F}) \cdot (1 - P(\text{BF} \mid \text{F}))^{f-j} \\ &= \binom{f}{j} \cdot P^j(\text{BF} \mid \text{F}) \cdot P^{f-j}(\text{LF} \mid \text{F}) \end{aligned} \quad (6.2)$$

for $0 \leq j \leq N(n+1)/2$ and $0 \leq f \leq N(3n+1)/2$.

6.3.1 Terms with Input or Output Stage Box Faults

Terms in the expression for $P(\text{loss} | N, f)$ can now be considered. The first term is considered first.

Theorem 6.1:

$$P(1 \leq b_n < f | N, f) = \sum_{j=1}^{f-1} P(b_n=j | N, f)$$

$$= \begin{cases} 0, & 0 \leq f \leq 1 \\ \sum_{j=2}^f \left[\frac{\sum_{b_n=1}^{j-1} \binom{N/2}{b_n} \binom{N_n/2}{j-b_n}}{\binom{N(n+1)/2}{j}} \right] * \binom{f}{j} * P^j(\text{BF} | F) * P^{f-j}(\text{LF} | F), & 1 < f \leq \frac{N}{2} \\ 1 - \sum_{j=0}^f \frac{\binom{N_n/2}{j}}{\binom{N(n+1)/2}{j}} * \binom{f}{j} * P^j(\text{BF} | F) * P^{f-j}(\text{LF} | F), & \frac{N}{2} < f \leq \frac{N_n}{2} \end{cases}$$

Proof: The case of $0 \leq f \leq 1$ is given for completeness.

Consider $1 < f \leq N/2$. Clearly,

$$P(1 \leq b_n < f | N, f) = \sum_{j=2}^f P(1 \leq b_n < j | N, b=j) * P(b=j | N, f)$$

for $1 < f \leq N/2$. From Equation 6.2,

$$P(b=j | N, f) = \binom{f}{j} * P^j(\text{BF} | F) * P^{f-j}(\text{LF} | F).$$

and for a fixed j ,

$$P(1 \leq b_n < j \mid N, b=j) = \sum_{b_n=1}^{j-1} P(b_n \mid N, b=j).$$

For a given b_n there are $\binom{N/2}{b_n} \binom{Nn/2}{j-b_n}$ possible fault configurations. The factor $\binom{N/2}{b_n}$ represents the number of ways b_n faults can be located in stage n boxes, while $\binom{Nn/2}{j-b_n}$ enumerates the ways $j-b_n$ box faults can fall outside of stage n . Note that $\binom{N(n+1)/2}{j}$ denotes the number of ways j box faults can occur in the ESC network without constraint. So,

$$P(b_n \mid N, b=j) = \frac{\binom{N/2}{b_n} \binom{Nn/2}{j-b_n}}{\binom{N(n+1)/2}{j}}.$$

Thus,

$$P(1 \leq b_n < f \mid N, f) = \sum_{j=1}^f \left[\frac{\sum_{b_n=1}^{j-1} \binom{N/2}{b_n} \binom{Nn/2}{j-b_n}}{\binom{N(n+1)/2}{j}} \right] * \binom{f}{j} * P^j(\text{BF} \mid F) * P^{f-j}(\text{LF} \mid F).$$

For $N/2 < f \leq Nn/2$ it is always the case that $b_n < f$. Thus, any pattern of faults with at least one fault in stage n contributes to this case. It is easier to count those fault distributions which do not contribute, i.e., those with $b_n = 0$. Formally,

$$P(1 \leq b_n < f \mid N, f) = 1 - \sum_{j=0}^f P(b_n=0 \mid N, b=j) * P(b=j \mid N, f)$$

for $N/2 < f \leq Nn/2$. There are $\binom{Nn/2}{j}$ ways j box faults can all occur outside

of stage n . So,

$$P(b_n=0 | N, b=j) = \frac{\binom{Nn/2}{j}}{\binom{N(n+1)/2}{j}}$$

Thus,

$$P(1 \leq b_n < f | N, f) = 1 - \sum_{j=0}^f \frac{\binom{Nn/2}{j}}{\binom{N(n+1)/2}{j}} * \binom{f}{j} * P^j(\text{BF} | F) * P^{f-j}(\text{LF} | F)$$

for $N/2 < f \leq Nn/2$.

□

Corollary 6.1: $P(1 \leq b_0 < f | N, f) = P(1 \leq b_n < f | N, f)$.

Proof: The corollary follows from the structural symmetry of the ESC.

□

Theorem 6.1 and Corollary 6.1 address the first two terms of the equation for $P(\text{loss} | N, f)$, respectively. The next theorem considers the third term of that equation.

Theorem 6.2:

$$P(b_n \geq 1 \text{ and } b_0 \geq 1 \mid N, f)$$

$$= \begin{cases} 0, & 0 \leq f \leq 1 \\ \sum_{j=2}^f \frac{\sum_{b_n=1}^j \sum_{b_0=1}^{j-b_n} \binom{N/2}{b_n} \binom{N/2}{b_0} \binom{N(n-1)/2}{j-(b_n+b_0)}}{\binom{N(n+1)/2}{j}} * \binom{f}{j} * P^j(\text{BF} \mid F) * P^{f-j}(\text{LF} \mid F), & 1 < f \leq N \end{cases}$$

Proof: The case for $0 \leq f \leq 1$ is trivial. Consider now $1 < f \leq N$.

$$P(b_n \geq 1 \text{ and } b_0 \geq 1 \mid N, f) = \sum_{j=2}^f P(b_n \geq 1 \text{ and } b_0 \geq 1 \mid N, b=j) * P(b=j \mid N, f).$$

From Equation 6.2

$$P(b=j \mid N, f) = \binom{f}{j} * P^j(\text{BF} \mid F) * P^{f-j}(\text{LF} \mid F).$$

For a fixed j ,

$$P(b_n \geq 1 \text{ and } b_0 \geq 1 \mid N, b=j) = \sum_{i=2}^j P(b_n \geq 1, b_0 \geq 1, \text{ and } b_n + b_0 = i \mid N, b=j).$$

The expression $\binom{N/2}{b_n}$ is the number of ways b_n box faults can be located in stage n . $\binom{N/2}{b_0}$ is the analogous expression for b_0 , and $\binom{N(n-1)/2}{j-(b_n+b_0)}$ enumerates the ways $j-(b_n+b_0)$ box faults can occur in stages $n-1$ through 1. Now, given the constraints $b_n \geq 1$, $b_0 \geq 1$, and $b_n + b_0 \leq j$ there are

$$\sum_{b_n=1}^1 \sum_{b_0=1}^{j-b_n} \binom{N/2}{b_n} \binom{N/2}{b_0} \binom{N(n-1)/2}{j-(b_n+b_0)}$$

possible fault configurations. The limits on the double summation are such that all possible b_n and b_0 values meeting the constraints are generated. The number of ways j box faults can be placed in the network without constraint is $\binom{N(n+1)/2}{j}$. So,

$$P(b_n \geq 1 \text{ and } b_0 \geq 1 | N, b=j) = \frac{\sum_{b_n=1}^1 \sum_{b_0=1}^{j-b_n} \binom{N/2}{b_n} \binom{N/2}{b_0} \binom{N(n-1)/2}{j-(b_n+b_0)}}{\binom{N(n+1)/2}{j}}$$

Thus,

$$P(b_n \geq 1 \text{ and } b_0 \geq 1 | N, f) = \sum_{j=2}^f \left[\frac{\sum_{b_n=1}^1 \sum_{b_0=1}^{j-b_n} \binom{N/2}{b_n} \binom{N/2}{b_0} \binom{N(n-1)/2}{j-(b_n+b_0)}}{\binom{N(n+1)/2}{j}} \right]^*$$

$$\binom{f}{j} \cdot P^j(\text{BF} | F) \cdot P^{f-j}(\text{LF} | F)$$

□

The fourth term of the general equation for $P(\text{loss} | N, f)$,

$$P(b_n + b_0 = 0 \text{ and loss} | N, f)$$

is more difficult to evaluate than the first three. In the next section it is analyzed for $f = 2$ and the results combined with the work of this section, evaluated for $f = 2$, to yield $P(\text{loss} | N, 2)$.

6.3.2 Term without Input or Output Stage Box Faults

Not all fault groupings satisfying the constraint $b_n + b_0 = 0$ result in loss of fault-free interconnection capability. This is a direct reflection of the robustness of the ESC in the face of multiple faults, and so is a positive attribute. Determining the fourth term of the equation for $P(\text{loss} | N, 2)$ is, consequently, involved.

A *lossy pair* is a *fault pair*, or pair of faults, that causes a loss of fault-free interconnection capability. A *BB-fault pair* (box-box fault pair) is a fault pair consisting of two faulty interchange boxes. An *LB-fault pair* and an *LL-fault pair* involve a faulty link and box, and two faulty links, respectively. The terms "LB-fault pair" and "BL-fault pair" are interchangeable. Lossy pairs can be analogously classified as *BB-lossy pairs*, *LB-lossy pairs*, and *LL-lossy pairs*. The number of lossy pairs in an ESC network is a function of N . Figure 6.2 shows an ESC network for $N=8$ with a faulty box and indicates by broken lines all network components that can form a lossy pair with the faulty box.

To determine the numbers of the types of lossy pairs, consider the sufficient conditions for a fault pair to be a lossy pair, or *lossy*. Let the two faults be in stages i and j , $1 \leq i, j \leq n$, and assume without loss of generality that $j \leq i$. If $(i, a_{n-1} \dots a_1 a_0)$ and $(j, b_{n-1} \dots b_1 b_0)$ are fault labels such that $a_{n-1} \dots a_{j+1} a_i = b_{n-1} \dots b_{j+1} b_j$ and $a_{j-1} \dots a_1 \bar{a}_0 = b_{j-1} \dots b_1 b_0$, then the fault labels denote a lossy pair since there will exist source/destination pairs with no fault-free connecting path (see Corollary 4.1).

Now consider the number of stage j faults which form a lossy pair with a given stage i fault. One case is when $i=j$. The constraints for loss of fault-free interconnection capability are then $a_{n-1} \dots a_{i+1} a_i = b_{n-1} \dots b_{i+1} b_i$ and

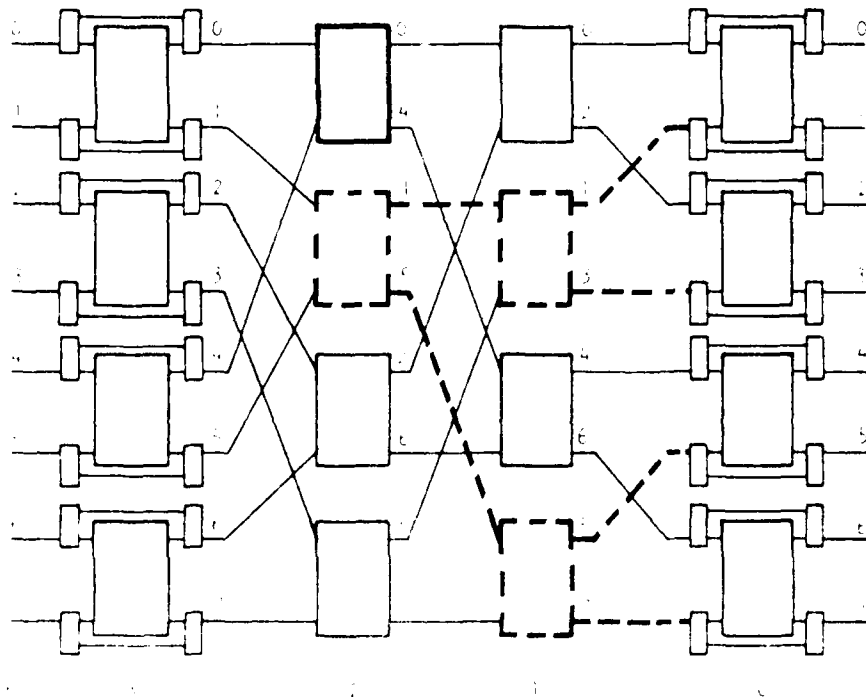


Figure 6.2 ESC network with $N=8$, a given faulty box (indicated by the bold line), and all network components in stages 2 and 1 with which it can form a lossy pair (indicated by the broken lines).

$a_{i-1} \dots a_1 \bar{a}_0 = b_{i-1} \dots b_1 b_0$. Only if the fault labels are complementary in bit position 0 and match in all other bit positions is such a fault pair ($i=j$) lossy. Thus, there is exactly one box and one link in stage i , $1 \leq i < n$, that can form a lossy pair with another given stage i fault. For $i=n$, there is one link that can form a lossy pair with another stage n link. This implies there are $N/4$ BB-lossy pairs in any given stage i , $1 \leq i < n$, because any given box in stages $n-1$ through 1 can form a lossy pair with one other of the $N/2$ boxes in its stage. Also, there are $N/4$ BL-lossy pairs in stage i , $1 \leq i < n$, for this case since each of the N links in a stage can form a lossy pair with one box in the same stage. Finally, there are $N/2$ LL-lossy pairs in stage i (including $i=n$) because any given link can form a lossy pair with one other of the N links in the same stage.

A second case is when $i \neq j$, and without loss of generality it is assumed that $i > j$. The constraints for lossy pairs are $a_{n-1} \dots a_{i+1} a_i = b_{n-1} \dots b_{i+1} b_i$ and $a_{j-1} \dots a_1 \bar{a}_0 = b_{j-1} \dots b_1 b_0$. In other words, bit positions $i-1$ through j are unconstrained, or need not match. These $i-j$ unconstrained bit positions allow 2^{i-j} boxes in stage j to form BB-lossy pairs with any given box in stage i , $1 \leq i < n$. Each box in stage i , $1 \leq i < n$, can also form a BL-lossy pair with $2 \cdot 2^{i-j}$ stage j links. The factor of two accounts for the two stage j links which can form a lossy pair with a stage i box for every stage j box that can form a lossy pair with that stage i box. There are 2^{i-j-1} stage j boxes that can form BL-lossy pairs with any given stage i link, $1 \leq i \leq n$. The minus one in the exponent is due to the fact that there are half as many stage j boxes which can form a lossy pair with a stage i link as can form a lossy pair with the stage i box associated with that link. (A stage i box has two associated stage i links.) Finally, 2^{i-j} stage j links can form LL-lossy pairs with any given stage i link.

$1 \leq i \leq n$, due to the i - j unconstrained bit positions.

Since $f = 2$ was assumed, there is only one fault pair in the network. Let the number of items of a type T be $\eta(T)$. For BB-lossy pairs

$$\eta(\text{BB-lossy pairs} | b_n + b_0 = 0, N) = \frac{N}{4}(n-1) + \sum_{i=2}^{n-1} \sum_{j=1}^{i-1} 2^{i+j} \frac{N}{2} \quad (6.3)$$

The first term on the right-hand side of this equation is the number of BB-lossy pairs in which both faults are in the same stage i , $1 \leq i < n$. Each term resulting from the double summation is the number of BB-lossy pairs for a given i and j , $i \neq j$. All possible pairs of i and j values, given the assumption $1 \leq j < i < n$, are included. This equation simplifies to

$$\eta(\text{BB-lossy pairs} | b_n + b_0 = 0, N) = \frac{1}{4} [2N^2 - 3Nn - N]$$

For LB-lossy pairs

$$\eta(\text{LB-lossy pairs} | b_n + b_0 = 0, N) = N(n-1) + \sum_{j=1}^{n-1} 2^{n-j-1} N + \quad (6.4)$$

$$\sum_{i=2}^{n-1} \sum_{j=1}^{i-1} (2 \cdot 2^{i+j} \frac{N}{2} + 2^{i+j-1} N)$$

The first term on the right-hand side of Equation 6.4 is the number of LB-lossy pairs that have both faults in the same stage for the $n-1$ stages in the ESC network (excluding stages n and 0). The single summation term denotes the LB-lossy pairs containing a stage n link; these are the only LB-lossy pairs that involve a stage n component. The double summation counts all remaining LB-lossy pairs. The first term inside the summation is the number of these pairs plus the faults in a stage n component and the last term is for those

LB-lossy pairs with a faulty stage i link. Equation 6.4 can be simplified, yielding

$$\eta(\text{LB-lossy pairs} \mid b_n + b_0 = 0, N) = 2N^2 - 2Nn - 2N.$$

The number of LL-lossy pairs,

$$\eta(\text{LL-lossy pairs} \mid b_n + b_0 = 0, N) = \frac{Nn}{2} + \sum_{i=2}^n \sum_{j=1}^{i-1} 2^{i+j} N. \quad (6.5)$$

The first term represents those lossy pairs of this type with both faults in the same stage, (i.e., $i = j$, $1 \leq i \leq n$). Each term of the double summation denotes the number of LL-lossy pairs for a given i and j , $i \neq j$. The expression simplifies as follows

$$\eta(\text{LL-lossy pairs} \mid b_n + b_0 = 0, N) = 2N^2 - \frac{3Nn}{2} - 2N.$$

At this point the number of BB-lossy pairs, LB-lossy pairs, and LL-lossy pairs, all not involving any stage n or 0 box, is known. Let α represent " $b_n + b_0 = 0$ " for conciseness in the statement of the following lemma.

Lemma 6.1

$$\begin{aligned} 1 - P(\text{loss} \mid \alpha, N \text{ BB-fault pair } (f=2)) &= \frac{\eta(\text{BB-lossy pairs} \mid \alpha, N)}{\eta(\text{BB-fault pairs} \mid \alpha, N)} \\ &= \frac{\frac{1}{4} [2N^2 - 3Nn - N]}{\binom{N(n-1)/2}{2}}. \end{aligned}$$

$$\begin{aligned}
 2. \quad P(\text{loss} \mid \alpha, N, \text{LB-fault pair } (f=2)) &= \frac{\eta(\text{LB-lossy pairs} \mid \alpha, N)}{\eta(\text{LB-fault pairs} \mid \alpha, N)} \\
 &= \frac{2N^2 - 2Nn - 2N}{(Nn)(N(n-1)/2)}
 \end{aligned}$$

$$\begin{aligned}
 3. \quad P(\text{loss} \mid \alpha, N, \text{LL-fault pair } (f=2)) &= \frac{\eta(\text{LL-lossy pairs} \mid \alpha, N)}{\eta(\text{LL-fault pairs} \mid \alpha, N)} \\
 &= \frac{2N^2 - \frac{3Nn}{2} - 2N}{\binom{Nn}{2}}
 \end{aligned}$$

Proof. The number of BB fault pairs with no faulty boxes in stages n or 0 is $\binom{N(n-1)/2}{2}$, i.e. the number of ways to select two boxes from the $N(n-1)/2$ not in stages n or 0 . Equation 6.3 gives the number of BB-lossy pairs with $b_n + b_0 = 0$. The number of LB-fault pairs not containing a stage n or 0 fault is $(Nn) \cdot \frac{N(n-1)}{2}$. There are Nn links, any one of which can be chosen independently of any one of $N(n-1)/2$ boxes. Equation 6.4 gives the corresponding number of LB-lossy pairs. Finally, LL-fault pairs number $\binom{Nn}{2}$ and Equation 6.5 enumerates LL-lossy pairs.

□

Lemma 6.2 Given two faults in an ESC network

$$P(b_n + b_0 = 0 | N, 2, \text{BB-fault pair}) = \frac{\binom{N(n-1)/2}{2}}{\binom{N(n+1)/2}{2}}$$

$$P(b_n + b_0 = 0 | N, 2, \text{LB-fault pair}) = \frac{\binom{N(n-1)/2}{1} N_n}{\binom{N(n+1)/2}{1} N_n}$$

$$P(b_n + b_0 = 0 | N, 2, \text{LB-fault pair}) = 1.$$

Proof. For each fault type the above equations enumerate the ways a given fault pair type can occur and satisfy the constraint $b_n + b_0 = 0$, divided by the number of ways it can occur. For LB-fault pairs only the faulty box is relevant to determining the value of $b_n + b_0 = 0$.

□

The goal of this section is to obtain an expression for $P(b_n + b_0 = 0 \text{ and loss} | N, 2)$ so $P(\text{loss} | N, 2)$ can be evaluated.

Theorem 6.1 Given two faults in an ESC network

$$P(b_n + b_0 = 0 \text{ and loss} | N, 2) = \sum_{\substack{\text{fault} \\ \text{pair} \\ \text{type}}} P(b_n + b_0 = 0, \text{loss, fault pair type} | N, 2)$$

$$\begin{aligned}
&= \frac{\frac{1}{4}[2N^2 - 3Nn - N]}{\binom{N(n-1)/2}{2}} * \frac{\binom{N(n-1)/2}{2}}{\binom{N(n+1)/2}{2}} * P^2(\text{BF} | F) + \\
&\frac{2N^2 - 2Nn - 2N}{(Nn)(N(n-1)/2)} * \frac{\binom{N(n-1)/2}{1}}{\binom{N(n+1)/2}{1}} * 2 * P(\text{BF} | F) * P(\text{LF} | F) + \\
&\frac{2N^2 - \frac{3Nn}{2} - 2N}{\binom{Nn}{2}} * P^2(\text{LF} | F).
\end{aligned}$$

Proof: Each of the three summation terms can be written $P(\text{loss} | b_n + b_0 = 0, \text{fault pair type}, N, 2)$ * $P(b_n + b_0 = 0 | \text{fault pair type}, N, 2)$ * $P(\text{fault pair type} | N, 2)$. Lemma 6.1 gives the first factor of this expression, and Lemma 6.2 the second. The third term can be computed using the relationship

$$P(x \text{ box faults} | N, 2) = \binom{2}{x} * P^x(\text{BF} | F) * P^{2-x}(\text{LF} | F)$$

where $x = 2$ for BB-fault pairs, $x = 1$ for LB-fault-pairs, and $x = 0$ for LL-fault pairs

□

6.3.3 Solution for Two Faults

Theorem 6.1, Corollary 6.1, Theorem 6.2, and Theorem 6.3 give expressions for each of the terms of $P(\text{loss} | N, 2)$, with parameters N , $P(\text{BF} | F)$, and $P(\text{LF} | F)$.

Theorem 6.4: Given two faults in an ESC network

$$\begin{aligned}
 P(\text{loss} | N, 2) = & \left[\frac{(4Nn - 2N) + (4N - 6n - 2)}{N(n+1)^2 - 2(n+1)} \right] * P^2(\text{BF} | F) + \\
 & \left[\frac{2Nn + (4N - 4n - 4)}{Nn^2 + Nn} \right] * 2 * P(\text{BF} | F) * P(\text{LF} | F) + \\
 & \left[\frac{4N - 3n - 4}{Nn^2 - n} \right] * P^2(\text{LF} | F)
 \end{aligned}$$

Proof. The result follows directly from the substitution into Equation 6.1 of terms derived from Theorems 6.1, 6.2, 6.3, and Corollary 6.1, given $f = 2$.

□

The coefficients of the equation for $P(\text{loss} | N, 2)$ in Theorem 6.4 have been verified by direct enumeration of lossy pairs for $N = 4, 8, 16, 32$, and 64 . See Appendix A for the computer program used.

6.4 Conclusions

This chapter has studied the multiple-fault tolerance of the ESC. Theorem 6.4 gave an expression for the probability of loss of fault-free interconnection capability given two faults, box and/or link, in the network.

These results will be pursued further in the next chapter where they will motivate a modification to the ESC to enhance its fault tolerance.

CHAPTER 7

PROPERTIES OF AN ENHANCED EXTRA STAGE CUBE INTERCONNECTION NETWORK

7.1 Introduction

The fault model of the ESC presupposes that the input demultiplexers and output multiplexers are fault-free. This is necessary in the ESC topology so that devices using the network will actually have access to the network. In this chapter a way to remove that constraint is given.

A further advantage of studying the multiple-fault tolerance of the ESC, as was done in the previous chapter, is that it may shed light on ways to improve the ability of the ESC to retaining fault-free interconnection capability in the face of multiple faults. The analysis is carried forward to show that input and output stage boxes are a significant source of all lossy pairs. A method of alleviating this effect is given; it can improve multiple-fault tolerance of the ESC significantly.

7.2 Eliminating Hardcore

The *hardcore* of a fault-tolerant system is components which are assumed to be fault-free. For the ESC the hardcore is the input demultiplexers and output multiplexers and their connections to the devices using the network. If any of these fails then the device attached to the network at that port no longer has access to the network and cannot communicate with (i.e., cannot

send messages to or cannot receive messages from) any other device.

If devices attached to the ESC each had two input/output ports then ESC input demultiplexers and output multiplexers would be unnecessary. One output of each device would be connected directly to a stage n box input, the other directly to the corresponding stage n multiplexer. Similarly, one device input would connect directly to a stage 0 box output, the other to the appropriate stage 0 demultiplexer. The advantage of this approach is that a single failure will not deny a device access to the network. Figure 7.1 shows an ESC network configured in this manner.

7.3. Individual Box Enabling/Disabling

Theorem 6.4 enumerates all lossy pairs, given that stages n and 0 are each bypassed in their entirety if required. Those lossy pairs that include a stage n or 0 box fault can be separated for comparison with those that do not. (The terms of Theorem 6.4 are not fully reduced so as to clarify this separation.) For BB-lossy pairs the ratio of pairs that include a stage n and/or 0 fault to those that do not is

$$\frac{4n - 2N}{4N - 6n - 2}$$

For LB-lossy pairs this ratio is

$$\frac{2n - 1}{4N - 4n - 1}$$

LB-lossy pairs cannot include stage n or 0 box faults.

Table 7.1 gives the value of these ratios for various values of N , and the percentage of lossy pairs of the two types involving stage n or 0 box faults. It

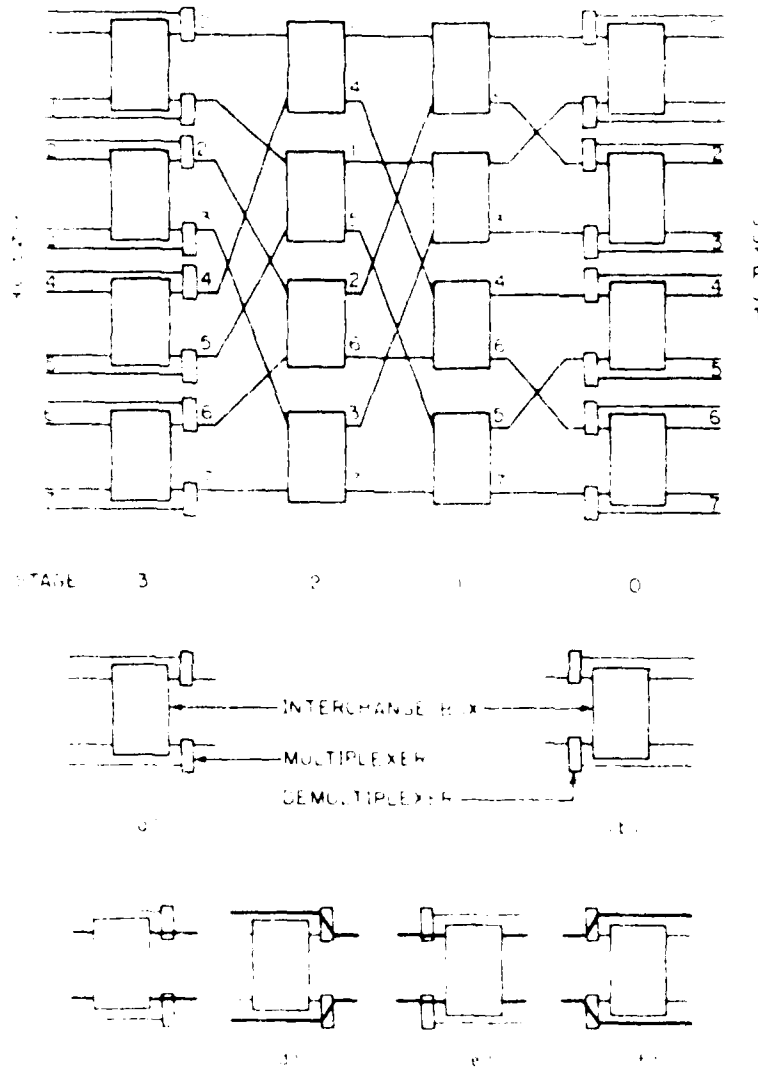


Figure 7.1 ESC network configured with two input and two output ports per device. (a) and (b) Detail of bypass circuitry. (c) and (d) Enabled and disabled input box, respectively. (e) and (f) Enabled and disabled output box, respectively.

Table 7.1. Ratio of lossy pairs involving a stage n or 0 box fault to those not, and the corresponding percentage of lossy pairs involving a stage n or 0 box fault, given stage bypassing.

N	BB-lossy pair ratio	Percent	LB-lossy pair ratio	Percent
4	12.00	92.3	4.00	80.0
8	6.65	86.9	3.00	75.0
16	5.89	85.5	2.91	74.4
32	5.00	83.7	3.08	75.5
64	6.46	86.6	3.37	77.1
128	7.11	87.7	3.73	78.9
256	7.89	88.8	4.15	80.6
512	8.74	89.7	4.59	82.1
1024	9.65	90.1	5.05	83.5
2048	10.5	90.5	5.53	84.7
4096	11.5	91.0	6.02	85.8

shows that with stage bypassing the majority of BB-lossy pairs contain stage n or 0 box faults. Further,

$$\lim_{N \rightarrow \infty} \frac{4Nn - 2N}{4N - 6n - 2} = \infty.$$

So, as network size grows more and more BB-lossy pairs include a stage n or 0 box fault. For LB-lossy pairs

$$\lim_{N \rightarrow \infty} \frac{2Nn}{4N - 4n - 4} = \infty.$$

The data presented in Table 7.1 should not be too surprising. Clearly, the majority of BB- and LB-lossy pairs contain stage n or 0 box faults. Bypassing all of stage n or 0 given a single fault is overly aggressive, especially as it guarantees any subsequent fault will complete a lossy pair. Bypassing only faulty boxes in stage n and 0 can improve ESC multiple-fault tolerance.

There are several ways to support box bypassing. One method is to include address decoding logic with each box disabling circuit. A system control unit would use an address bus to command the multiplexers and demultiplexers of the failed box to the disable state. This scheme requires $N/2$ such decoding circuits and an $n+1$ bit bus for both stage n and 0.

One is to dedicate a control line for the bypass circuitry with each box, instead of one per stage; no new logic circuitry in the ESC is required for this method. There are two ways these lines can be used. In one, the lines are connected to a system control unit, and it activates the appropriate line. This scheme is likely to be cumbersome for large networks. An alternative is to link each line to the device using the box it controls. When a stage n or 0 box fault occurs a special fault label is passed by a system control unit to all devices using the network. Devices are responsible for determining the correct state for

boxes under their control. If the network is used in a unidirectional mode, each device is responsible for both a stage n and 0 box.

Operational protocol with box bypassing is as follows. The fault-free network configuration is the same as with stage bypassing. Given f faults, $1 \leq f$, if $b_n = f$ then all stage n boxes are bypassed and all stage 0 boxes are enabled; if $b_0 = f$ then all stage 0 boxes are bypassed and all stage n boxes are enabled; if neither of the preceding conditions is true then every faulty stage n and 0 box is bypassed, and the remaining stage n and 0 boxes are enabled. Note that box faults not in stages n or 0 and link faults may be overcome due to the two paths from each source to any destination available by enabling all boxes of stages n and 0, as would be the case with stage bypassing.

The ESC using box bypassing can pass Generalized Cube performable permutations in at most two passes despite faults, provided fault-free interconnection capability is retained, just as when using stage bypassing. To accomplish this, stage bypassing is simulated with box bypassing, and the procedure is the same used when the network is operated with stage bypassing.

If the routing tag method is to be applied for determining if the network retains fault-free interconnection capability, it can readily be extended for use with box bypassing. Routing tag computation is as follows. When there are no faults, the routing tags and broadcast routing tags as specified in Tables 4.2 and 4.3 for the no-fault situation are used. If the fault location is a stage 0 box then the special fault label is used to determine if the primary path contains the fault stage. The result is, if so, then the tag for a stage 0 fault (Tables 4.2 and 4.3), i.e., 1. If the fault is in stage i , $0 < i < n$, or any link, then the primary path is used if it is fault-free, otherwise, the secondary path is taken. Finally, if the stage n box to which a device is

connected is bypassed then the device uses tags formed as if stage n were bypassed, i.e., the fault location is a stage n box.

7.4 Reliability of the Enhanced Extra Stage Cube Network

The value of $P(\text{loss} | N, 2)$ can now be evaluated directly in terms of fault pairs. Removing the restriction to only stages $n-1$ through 1 adds $\frac{3N^2}{2} - 2N$ new BB-lossy pairs to the total given by Equation 6.3, $2N^2 - 2N$ new LB-lossy pairs beyond those counted in Equation 6.4, and no new LL-lossy pairs. The new BB-lossy pairs include $N(2^{n+1}-2)/2$ involving a stage n box fault, and $N(2^n-2)/2$ involving a stage 0 box fault and no stage n box fault. The new LB-lossy pairs include $N(2^{n+1}-2)/2$ involving a stage n box fault, and an equal number with a stage 0 box fault. The method used to count the additional lossy pairs is analogous to the methodology used in Chapter 6.

Theorem 7.1: Given two faults in an ESC network using box bypassing

$$\begin{aligned}
 P(\text{loss} | N, 2) = & \left[\frac{14N - 6n - 18}{N(n+1)^2 - 2(n+1)} \right] * P^2(\text{BF} | F) + \\
 & \left[\frac{8N - 4n - 8}{(Nn^2 + Nn)} \right]^2 * P(\text{BF} | F) * P(\text{LF} | F) + \\
 & \left[\frac{4N - 3n - 4}{Nn^2 - n} \right] * P^2(\text{LF} | F)
 \end{aligned}$$

Proof With box bypassing,

$$P(\text{loss} | N, 2) = \sum_{\substack{\text{fault} \\ \text{pair} \\ \text{types}}} \frac{\eta(\text{lossy pair type} | N)}{\eta(\text{fault pair type} | N)} * P(\text{fault pair type}),$$

because lossy pairs are now defined for all stages of the network. Adding the newly resulting lossy pairs to those enumerated in their respective equations (Equations 6.3, 6.4, and 6.5), and using the resulting values in equations analogous to those of Lemma 6.1, but including all network stages, yields the terms of the above summation. □

Table 7.2 gives the ratios of probability of loss of fault-free interconnection capability using stage bypassing to using box bypassing, for BB- and LB-lossy pairs. For example, Table 7.2 shows that for $N = 1024$ an ESC network with box bypassing is 2.63 times less likely to lose fault-free interconnection capability due to a BB-fault pair than is an ESC with stage bypassing. For LB-fault pairs the ratio is 6.02. Of course, additional ESC circuitry (if any) used to implement box bypassing can fail in an actual installation, decreasing reliability.

The fault tolerance improvement with box bypassings depends on the values of $P(BE_i | E_i)$ and $P(LE_i | E_i)$. These values can be based on assumptions about particular implementations or experiments with actual hardware. The values determined for $P(BE_i | E_i)$ and $P(LE_i | E_i)$ can be inserted into the equations of Theorems 6.4 and 7.1, and the respective values for $P(\text{loss} | N, 2)$ calculated to determine the improvement due to box bypassing. As an example, consider $P(BE_i | E_i) = P(LE_i | E_i)$. For $N = 1024$, $P(\text{loss} | N = 1024, 2) = 0.919$ using stage-

Table 7.2 Ratio of probability of loss of fault-free interconnection capability using stage bypassing versus box bypassing, for BB- and LB-lossy pair types. The probability of loss of fault-free interconnection capability due to LL-lossy pairs does not depend on bypassing method.

N	BB-lossy pair ratio stage bypassing to box bypassing	LB-lossy pair ratio stage bypassing to box bypassing
4	0.765	2.50
8	1.00	2.36
16	1.22	3.31
32	1.45	3.72
64	1.68	4.15
128	1.91	4.60
256	2.15	5.06
512	2.39	5.54
1024	2.63	6.02
2048	2.88	6.51
4096	3.13	7.00

bypassing (from Theorem 4), and $P(\text{loss} | N=1024, 2) = 0.260$ using box bypassing, so the improvement is by a factor of 3.5. Two special cases are $P(\text{LF} | F) = 1$ and $P(\text{BF} | F) = 1$. If $P(\text{LF} | F) = 1$, i.e., $P(\text{BF} | F) = 0$, then there is no fault-tolerance improvement; either bypassing policy will enable all stage n and 0 boxes in the event of a fault or faults. If $P(\text{BF} | F) = 1$ only BB-fault pairs are possible, and the first column of Table 7.2 shows the factor by which $P(\text{loss} | N, 2)$ would decrease.

The coefficients of the equation for $P(\text{loss} | N, 2)$ with box bypassing have been verified by direct enumeration for $N = 4, 8, 16, 32$, and 64. See Appendix B for the computer program used. The dashed lines in Figure 7.2 show $P(\text{loss} | N, 2)$ with box bypassing for $4 \leq N \leq 8192$ and a range of $P(\text{BF} | F)$ values. Note that the dashed line for $P(\text{BF} | F) = 0$ is coincident with the solid line (stage bypassing) for the same $P(\text{BF} | F) = 0$.

7.6 Conclusions

This chapter described a modification for the ESC that improves its multi-fault tolerance. Rather than bypass the entire stage if a stage n or 0 fails, only the failed box is bypassed. Little or no additional network hardware is required, depending on the implementation method chosen, to support this method of operation.

The fault tolerance of the ESC to two faults was analyzed so that the extent of box bypassing could be determined. This analysis indicated that box bypassing can provide greater tolerance to two faults. The fault-tolerance improvement was quantified. For example, at 1.3% with $N = 1024$ using box bypassing is 2.5 times less likely to lose a fault-tolerant computation capability due to two faults than if using stage bypassing assuming box and link faults.

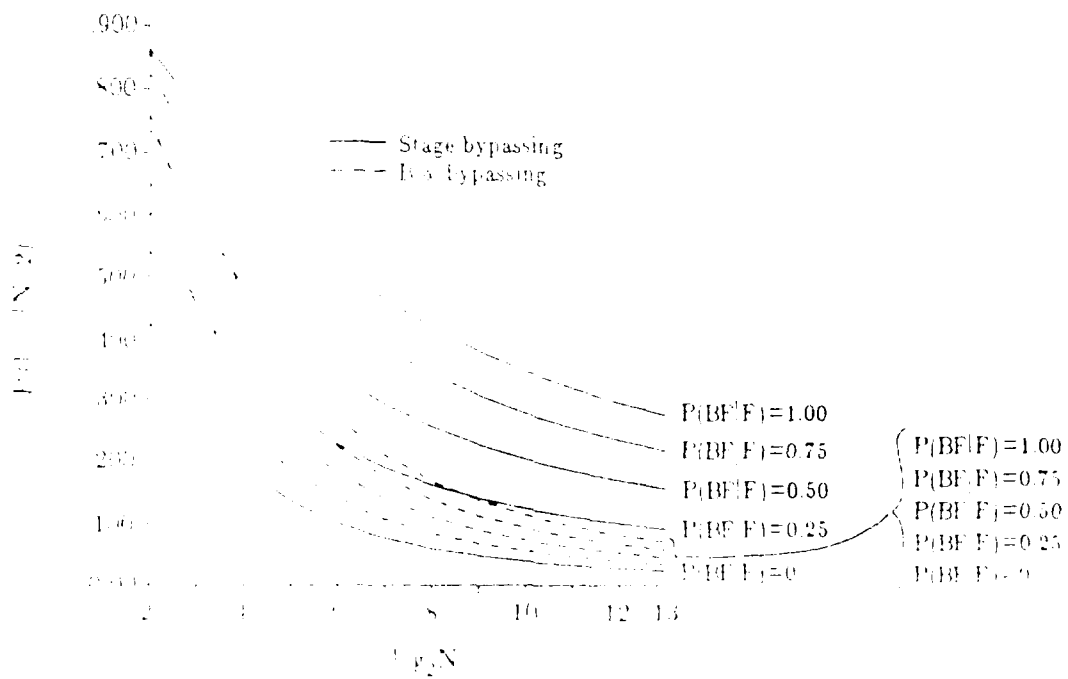


Figure 7.2 $P(\text{loss}|N, 2)$ as a function of network size for various values of $P(\text{BF}|F)$. Solid lines are for stage bypassing; dashed lines are for box bypassing. The solid lines in Figure 7.2 show $P(\text{loss}|N, 2)$ for $4 < N \leq 8192$ and a range of $P(\text{BF}|F)$ values.

are equally likely. The fact that large networks gain more than smaller ones underscores the value of this modification for use in large-scale parallel processing systems: larger networks are more likely to experience a fault at any given time.

CHAPTER 8

ANALYSIS OF ALTERNATIVE SWITCHING ELEMENTS

8.1 Introduction

After basic capability has been established, a next step in the engineering of a new interconnection network is a consideration of its implementation. The advent of Very Large Scale Intergration (VLSI) technology motivates the use of complex building blocks in constructing a system, especially when the number of distinct parts can be kept small. In recognition of the growing applicability and capability of VLSI technology, many papers have included discussions on using switching elements larger than 2×2 interchange boxes for implementing multistage cube-type networks [AdS84b, ChM82, CiS81, CiS80, HuM81, Law75, MAS81, Pat81, Pea77, PKM80, Smi81]. The ESC was defined in terms of interchange boxes, bypass circuitry, and the connections between them, thus, the ESC can be constructed from just a few different parts. Interchange box circuitry requirements do not tax the current state of the VLSI art (pinout requirements are somewhat more *challenging*).

An interchange box is a 2-input/2-output crossbar switch. A more general form of interchange box is an a -input/ a -output ($a \times a$) crossbar. A network closely related to the ESC can be constructed from $a \times a$ switching elements, $a = 2^i$ for $i = 2, 3, 4, \dots$, using cube interconnections between stages. This chapter examines the relative merits of 2×2 and 4×4 crossbar switching elements, or nodes, with regard to performance. A 4×4 crossbar node is a

logical successor to the interchange box, yet it is not so complex as to preclude implementation on a single VLSI chip [MAS81].

Thus, this chapter addresses one high level issue confronting a network designer: choice of switching element architecture to satisfy a performance goal. Other design goals such as cost, power consumption, testability, and reliability are also aspects of any development project and will mandate switching element structure. Other high level issues such as choice of integrated circuit technology and packaging constraints will influence the choice of structure. These issues would be decided in conjunction with or subsequent to the investigation carried out here. Beyond these high level issues are low level issues such as details of circuit design, circuit layout, and design of test sequences. Concurrent and subsequent activity in such areas as design verification and simulation are also necessary to bring a concept into physical realization.

Following a review of existing literature on $a \times a$, $a > 2$, switching elements for use in multistage cube-type networks, the switching element model used for the performance analysis here is given. With this background the performance of 4×4 crossbar nodes and a 4-input/4-output node constructed from interchange boxes is compared. A method of constructing ESC networks from these switching elements and their effects on network performance is described.

8.2 Previous Work

Others have considered implementation of multistage cube-type networks using substitutes for interchange boxes (2×2 switches). In [Pea77], Pease notes that there is considerable flexibility when making implementation decisions concerning the indirect binary n -cube network. In [Law75], Lawrie mentions several design options for the Omega network. These networks are topologically equivalent to the Generalized Cube [SiS78], as mentioned in Chapter 3, Section 3.2. One possibility is to substitute 4×4 omega networks for a group of four 2×2 exchange-broadcast units. The functionality of an omega network built from these elements would be identical to one constructed from 2×2 elements. Another possibility mentioned is to construct the 4×4 elements with crossbar and broadcasting capabilities. In this case, the resulting network is more powerful than the basic omega network constructed from 2×2 elements.

The possibility of using crossbars of size $a \times a$, where $a = 2, 4$, or 8 to implement *digit controlled*, or *delta*, networks is discussed by Patel in [Pat81]. Due to technological limitations, Patel concludes that crossbars of size 8×8 are probably not practical at the present time, however, 4×4 crossbars are considered feasible. A performance analysis is presented for delta networks with 2×2 crossbar elements. The *probability of acceptance* is the likelihood that a processor request for connection through the network will be granted. *Bandwidth* is defined as the expected number of requests accepted per network cycle. The probability of acceptance of a request and the expected bandwidth of any $2^n \times 2^n$ delta network, given some mean request generation rate, is given in the paper. The results are extended to $a^n \times a^n$ delta networks.

The performance of buffered delta networks in a packet switched environment is considered by Dias and Jump in [DiJ81]. The omega network, a member of the class of delta networks, is analyzed in terms of throughput and turn-around-time. *Throughput* is defined as the average number of packets delivered in unit time for a particular network and environment. *Turn-around-time* is the average time interval between the time a packet is placed in a buffer at a network input and the time at which it is placed in a buffer at a network output. The analysis and simulations show that in terms of throughput, buffered delta networks compare favorably with unbuffered crossbar networks in a packet switched environment. The study is, however, carried out only for delta networks using 2×2 crossbar switching elements. As with the special case of omega networks, delta networks using 4×4 crossbar switching elements will be more powerful than those with 2×2 elements.

Cuniniera and Serra have investigated the probability of acceptance, given an asynchronous control scheme, of a class of cube-type networks using circuit switching [CS81]. Again, *acceptance* is defined as the immediate granting of a processor request for a connection through the network. The assumptions used are that processors generate random independent requests, uniformly distributed among all the memory modules of the system; requests are granted or rejected in negligible time, and only one request may be generated by a processor at a time. With this model the probability of acceptance is evaluated. The use of an asynchronous control scheme is a model which finds particular applicability in MIMD systems.

The class of SW-banyan networks is very general [GoL73, Lit77]. Switching structures of size $a \times b$ are possible, where a and b are positive integers. Franklin has compared banyan and crossbar networks [Fra81]

Performance is measured using a space-time measure obtained from a model incorporating physical VLSI implementation area requirements and network delay. Crossbar networks were found to have similar performance characteristics to Banyan networks with respect to this criterion. This measure is claimed to be particularly suited for quantifying network performance given the assumption of VLSI implementation.

8.3 Switching Element Model

The *Generalized Cube* operation model assumed for the analysis in this chapter includes *routing tags* [Law75, SiM81b] and *packet-switched* message handling [McS80]. That is, a packet consisting of a routing tag and a number of data items makes its way from stage to stage, releasing lines and switching nodes immediately after using them. The size of each input queue in a switching node is assumed to be an integral multiple of the packet size. Thus, packet size is not restricted to any particular number of words. Packet switching in multistage networks constructed from interchange boxes has been discussed in [DiS1, DiS9, McS80]. Packet switching in SW-banyan networks has been discussed in [Tr79].

The aim of this chapter is to investigate the performance of 4×4 crossbars versus 2×2 or 3×3 (interchange boxes). Since a single interchange box is not functionally comparable to a 4×4 crossbar (i.e., it can only handle two items at a time instead of four), the 4×4 crossbar is compared to a 4-input/4-output configuration of four interchange boxes. This configuration is called a *4-input/4-output node* and is shown in Figure 8.1. *Level 1* of a composite node is the two interchange boxes connected to the node inputs. *Level 2* consists of the two interchange boxes connected to the outputs. A Generalized Cube

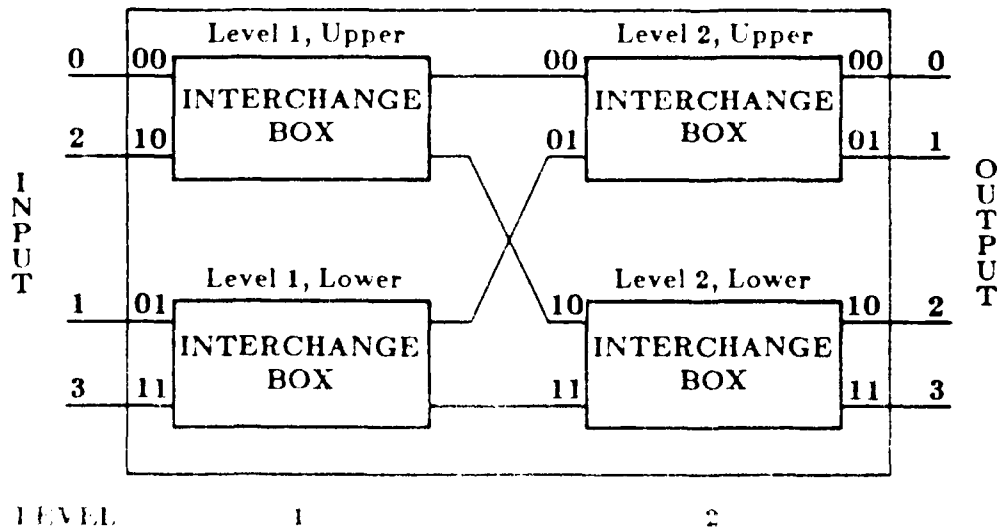
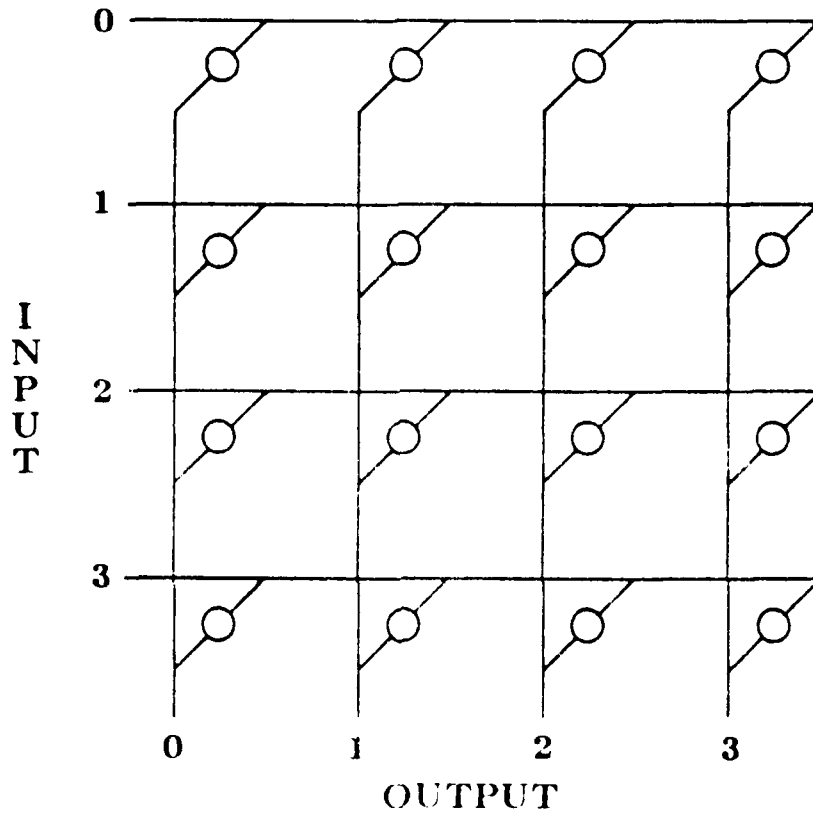


Figure 8.1 A 4x4 composite switching element, or node.

network constructed from properly connected (to be specified later) 4×4 composite nodes is identical to one constructed from interchange boxes. Examination of the 4×4 crossbar node in Figure 8.2(a) shows that its external connections are identical to those of the 4×4 composite node, so it can be directly substituted for a 4×4 composite node. Figure 8.2(b) depicts the *crosspoint*, or switch between an input and an output. Henceforth, both crossbar and composite nodes are assumed to be 4×4 , unless otherwise stated.

The performance of the crossbar node and the composite node will be compared on both a local and global level. On the local level, blocking within a node is examined. On the global level, the permuting ability of two networks constructed from the respective switching nodes is compared. Intuitively, one would expect that the crossbar nodes would be superior with respect to blocking and that networks constructed from them could perform more permutations. Indeed this is the case. The purpose of this section is to quantify these differences in performance. In the analysis for both the crossbar node and the composite node of the time required for messages to pass through the nodes, the following assumptions are made.

1. Initially, nodes are empty.
2. There can be from 0 to 4 messages at the inputs of a node at any one time.
3. Each message has only one destination (i.e., no broadcasting).
4. The desired node output for each message is a uniformly distributed random variable.
5. A message requires the same time to pass through either a 2×2 or 4×4



(a)



(b)

Figure 8-2 (a) A 4x4 crossbar switching element, or node (b) Crosspoint

crossbar. Hence, the minimum time to traverse a 4×4 crossbar node is half that of the 4×4 composite node.

The third assumption is made to simplify the analysis. The fourth assumption is equivalent to asserting that message destinations (network outputs) are a uniformly distributed random variable. This is because if each node output is selected with equal probability then there is an equal probability of reaching any given network output. The last assumption is based on the circuit characteristics of the two node designs discussed in [McM82]. Based on levels of logic it is reasonable to assume that the nodes would operate at similar speeds. However, if there is a speed difference, the performance analysis of the following sections would remain valid after the application of the appropriate weighting factor to the results. The last assumption simplifies the analysis.

8.4 4×4 Crossbar Node Performance Analysis

The time required for a single message to pass through a crossbar is referred to as a *time step*. Let t be the maximum number of messages destined for the same crossbar node output. Then t is the time, in units of time steps, required for all messages to transit the node, since the t contending, or conflicting, messages must exit the node sequentially and no other output node has a greater number of messages to pass. Let m represent the number of messages present at the inputs of a node. The probability that the time for all messages to transit a node is T , given that there are M messages, is denoted $P(t=T | m, M)$. The expected transit time given M messages is denoted $E(t | m, M)$.

If a single message is present at the inputs of a crossbar node it will pass through (transit) in one time step. Thus,

$$P(t=1 | m=1) = 1.$$

For $m=2$, transit times of one or two are possible. A transit time of two occurs only if the two messages contend for a given node output. There are four ways this can occur: one for each node output. Since message destinations are independent there are 4^m distinct choices of destinations. Thus, a transit time of one occurs for all 4^m choices except the four with contention, so

$$P(t=1 | m=2) = \frac{(16-4)}{4^2} = \frac{3}{4}$$

and

$$P(t=2 | m=2) = \frac{4}{4^2} = \frac{1}{4}.$$

So

$$E(t | m=2) = \sum_{i=1}^2 i * P(t=i | m=2) = 1 + \frac{1}{4}.$$

Note that

$$\sum_{i=1}^2 P(t=i | m=2) = \frac{3}{4} + \frac{1}{4} = 1$$

as required by the axioms of probability [Pap65]

For $m=3$, $1 \leq i \leq 3$, there are $4^3 = 64$ choices of destinations for the three messages. When there is only one message per output, there are

$4 \cdot 3 \cdot 2 = 24$ possible choices. Hence,

$$P(t=1 | m=3) = \frac{24}{4^3} = \frac{3}{8}.$$

If some output has two messages, then there are $\binom{3}{2} \cdot 4 \cdot 3 = 36$ choices. The factor $\binom{3}{2}$ is the number of message pairs which can be assigned to any of the four outputs. Recall that $\binom{a}{b} = \frac{a!}{(a-b)!b!}$. The remaining message can go to any of the three unused outputs. Thus,

$$P(t=2 | m=3) = \frac{36}{4^3} = \frac{9}{16}.$$

There are four ways three messages can contend for one output, requiring three time steps for all to transit the node. Hence,

$$P(t=3 | m=3) = \frac{4}{4^3} = \frac{1}{16}.$$

Therefore,

$$E(t | m=3) = \sum_{i=1}^3 i \cdot P(t=i | m=3) = 1 \frac{11}{16},$$

and

$$\sum_{i=1}^3 P(t=i | m=3) = \frac{3}{8} + \frac{9}{16} + \frac{1}{16} = 1,$$

as required.

For $m = 4$, $1 \leq t \leq 4$, there are $4!$ ways the messages can have distinct outputs. So,

$$P(t=1 | m=4) = \frac{4!}{4^4} = \frac{3}{32}.$$

There are two ways $t = 2$ can occur with $m = 4$. One is when only one output has two messages. There are $\binom{4}{2} \cdot 4 \cdot 3 \cdot 2$ ways this can occur. Another way is for two outputs to have two messages each. There are $\binom{4}{2} \cdot \binom{4}{2}$ ways to realize this; the reasoning is as follows. The first $\binom{4}{2}$ factor is the number of message pairs that can be formed by taking two messages from four. Furthermore, it is also the number of message pairs that can be formed by using all four messages, since choosing one pair implicitly creates a second. The second $\binom{4}{2}$ factor is the number of ways to choose two distinct outputs for the message pairs. Thus,

$$P(t=2 | m=4) = \frac{\left[\binom{4}{2} \cdot 4 \cdot 3 \cdot 2 + \binom{4}{2} \cdot \binom{4}{2} \right]}{4^4} = \frac{45}{64}.$$

For $t = 3$, one output must occur, shared by three messages. There are $\binom{4}{3} \cdot 4 \cdot 3$ ways this can occur. Hence,

$$P(t=3 | m=4) = \frac{\binom{4}{3} \cdot 4 \cdot 3}{4^4} = \frac{3}{16}.$$

Finally, there are four ways all messages can use the same output, thus

$$P(t=4 | m=4) = \frac{4}{4^4} = \frac{1}{64}.$$

Therefore,

$$E(t|m=4) = \sum_{i=1}^4 i \cdot P(t=i|m=4) = 2 \frac{1}{8},$$

and

$$\sum_{i=1}^4 P(t=i|m=4) = \frac{3}{32} + \frac{45}{64} + \frac{3}{16} + \frac{1}{64} = 1,$$

as required. If $P(m=i)$ is uniform over $1 \leq i \leq 4$, then

$$E(t) = \frac{1}{4} \sum_{i=1}^4 E(t|m=i) = 1 \frac{33}{64}.$$

The preceding analysis is most applicable to the operation of switching elements in an MIMD system. This is because the number of messages present at any given time on switching element inputs and message destinations were both assumed to be random, and in the case of destination addresses, uniformly distributed. (No assumption was made about the distribution of arriving messages.) These assumptions form a plausible model for MIMD computations, since such processing is characterized by communication that is, typically, weakly structured and of variable intensity.

Operation of a switching element in an SIMD environment will more usually entail permutation connections among the devices using the network. Consider the crossbar node in these circumstances (i.e., $m=4$ and each message is destined for a unique network output). The crossbar node is capable of performing all $4! = 24$ permutations of four items without conflict. While network outputs are unique for each message, network topology may require several messages to use the same output of some switching element. Thus, the probability that switching elements will introduce delay as a result of conflict is

of interest. If node output selection is random with uniform distribution, then for the crossbar node

$$P(\text{delay} | m=4) = 1 - P(\text{no conflict} | m=4) = 1 - \frac{4!}{4^4} = 0.906 .$$

Thus, 90.6 percent of the time when four messages try to pass through a crossbar node simultaneously there will be conflict, assuming random node output selection with uniform distribution. However, the permutations used frequently in a particular SIMD calculation [Len78] and the topology of the chosen interconnection network may result in node output needs that are far from uniformly distributed for any given node. Thus, $P(\text{delay} | m=4)$ may not be useful for understanding network level, as opposed to switching element level, behavior.

8.5 4×4 Composite Node Performance Analysis

The analysis of the composite node presented in this section is much less straightforward than that of the crossbar node. Despite the fact that performance is measured here in terms of time to pass information, the composite node analysis is presented as a case by case analysis of the ways in which conflict can occur in the node. The completeness and accuracy of the analysis is more apparent with this approach. Performance values can then easily be associated with each case for comparison of the composite node with the crossbar node.

8.5.1 Problem Overview and Notation

Table 8.1 lists all ways conflict can occur in the composite node. For each way the performance analysis cases are given. The table includes new terminology that is used in the composite node analysis. A *delay* occurs when a message arrives at, or requests arrival at, an interchange box input which already has one or more messages awaiting passage through the box. The new message must wait in turn for passage and is therefore delayed. A *subsequent delay* is simply a delay that occurs after an earlier delay or earlier conflict. Figure 8.3(a) shows an example of a subsequent delay due, in this instance, to an earlier conflict. In (1) messages A and B are shown in conflict. To resolve the conflict, A is arbitrarily selected, so (2) shows B ready to transit the box one time step later. The transit time for B is two time steps because of the delay subsequent to the initial conflict.

A *subsequent conflict* is analogous to a subsequent delay. Figure 8.3(b) shows a subsequent conflict. Initially, messages A and C conflict and message B requires use of the same box input as A. The conflict is resolved by arbitrarily choosing A, and (2) shows B and C at the box inputs after one time step. Because B happens to need the same box output as C (and A), B and C conflict. This conflict is subsequent to that of A and C.

An *arbitration delay* is a delay that occurs as a consequence only of the way a conflict in an interchange box is arbitrated. *Arbitrating*, or resolving a conflict means selecting one of the conflicting messages to pass first. The other message waits and, thus, may cause a delay for a third message. Changing the choice for resolving the conflict would eliminate that delay. Figure 8.3(c) shows an arbitration delay. In (1) messages A and C conflict, and B requires use of the same box input as A. After one time step and the selection of C to

Table 8.1 A complete listing of the ways conflict can occur in the 4×4 composite node as a function of m , the number of messages initially at the inputs of the node. The associated probability term is given after each table entry corresponding to a case of the performance analysis.

- I. $m=1$ (no conflict possible); $P(t=2 \mid m=1)$
- II. $m=2$
 - A. No conflict in level 1
 - 1. No conflict in level 2; $P(t=2 \mid m=2)$
 - 2. Conflict in level 2; $P(t=3 \mid m=2)$
 - B. Conflict in level 1 (cannot also have conflict in level 2); $P(t=3 \mid m=2)$
- III. $m=3$
 - A. No conflict in level 1
 - 1. No conflict in level 2; $P(t=2 \mid m=3)$
 - 2. Conflict in level 2; $P(t=3, \text{ case 2} \mid m=3)$
 - B. Conflict in level 1
 - 1. No conflict in level 2; $P(t=3, \text{ case 1} \mid m=3)$
 - 2. Conflict in level 2
 - a. One conflict only; $P(t=3, \text{ case 3} \mid m=3)$
 - b. One conflict and an arbitration delay; $P(t=4 \mid m=3)$
 - c. One conflict and a subsequent conflict; $P(t=4 \mid m=3)$

Table 8.1. continued.

IV. $m=4$

A. No conflict in level 1

1. No conflict in level 2; $P(t=2 \mid m=4)$
2. Conflict in level 2; $P(t=3, \text{ case 1} \mid m=4)$

B. Conflict in level 1

1. No conflict in level 2

- a. No conflict in level 2 given three messages enter level 2 initially; $P(t=3, \text{ case 2} \mid m=4)$
- b. No conflict in level 2 given two messages enter level 2 initially; $P(t=3, \text{ case 4} \mid m=4)$ and $P(t=3, \text{ case 5} \mid m=4)$

2. Conflict in level 2 given three messages enter level 2 initially

- a. One conflict only; $P(t=3, \text{ case 3} \mid m=4)$
- b. One conflict and an arbitration delay; $P(t=4, \text{ case 1} \mid m=4)$
- c. One conflict and a subsequent conflict; $P(t=4, \text{ case 1} \mid m=4)$

3. Conflict in level 2 given two messages enter level 2 initially

- a. No conflict for first two messages entering level 2, but conflict for the second two messages; $P(t=4, \text{ case 3} \mid m=4)$
- b. Conflict for the first two messages entering level 2 only; $P(t=4, \text{ case 2} \mid m=4)$
- c. Conflict for the first two messages entering level 2, and a subsequent conflict; $P(t=4, \text{ case 4} \mid m=4)$
- d. Conflict for the first two messages entering level 2, a subsequent conflict, and an arbitration delay; $P(t=5 \mid m=4)$
- e. Conflict for the first two messages entering level 2, a subsequent conflict, and another subsequent conflict; $P(t=5 \mid m=4)$

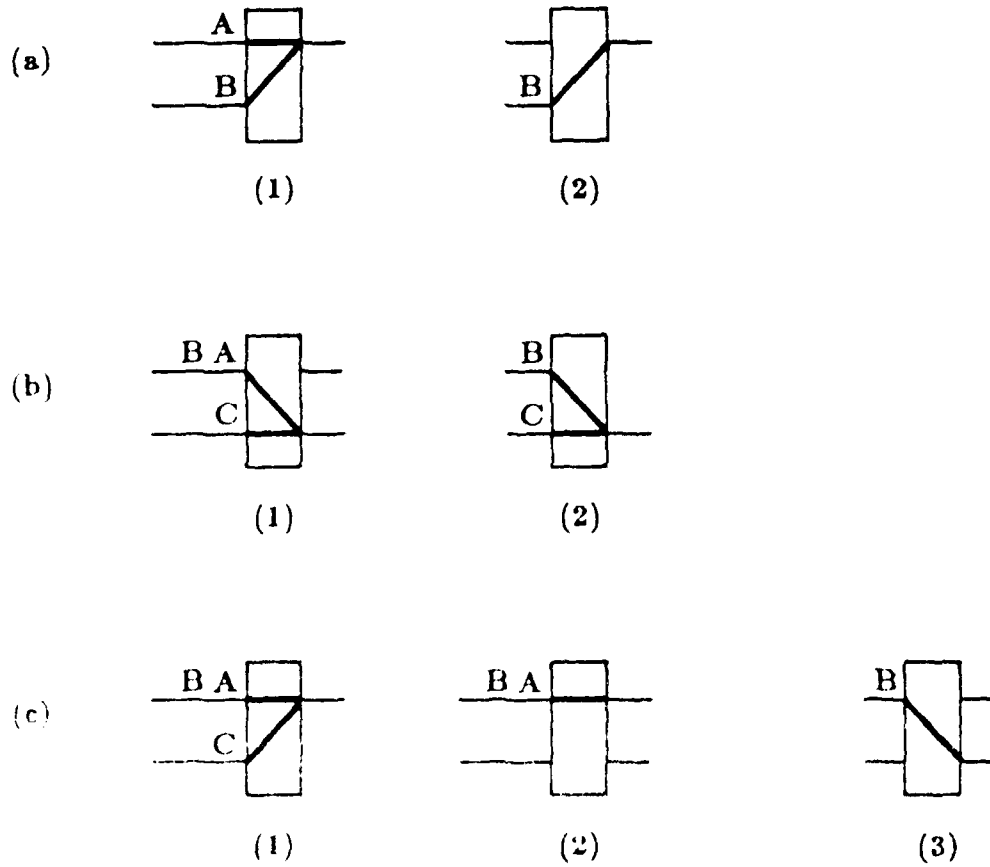


Figure 8.3 (a) Example of a subsequent delay. (b) Example of a subsequent conflict. (c) Example of an arbitration delay.

resolve the conflict, the messages are as depicted in (2). Message B must wait one more time step for A to clear the box before it can pass, even if B needs a different box output than A, as shown in (3). Note that if A were chosen to resolve the initial conflict, B and C would not subsequently conflict in this example. Thus, regardless of the box output need of B, it is delayed as a consequence of the resolution of the conflict between A and C. Note that an arbitration delay is a special type of subsequent delay. A conflict is never the result only of arbitration because the two messages involved in a conflict must need the same interchange box output.

Two examples follow to illustrate how the entries of Table 8.1 are determined. First, consider the situation depicted by Figure 8.4, which corresponds to table entry IV.B.2.b. Figure 8.4(a) shows the initial message positions and the level 1 box connections chosen to satisfy the conditions of this table entry. The upper box of level 1 has a conflict. This meets the first condition (a conflict in level 1) of IV.B.2.b. Next, Figure 8.4(b) shows the message locations in the node and updated box connection requests after the first time step. Message A was chosen arbitrarily in the first time step to resolve the conflict in the upper level 1 box. At this point a conflict exists between A and C on level 2, satisfying the second condition of IV.B.2.b. The connection request shown for message D is arbitrary; it is not affected by the conditions defining IV.B.2.b case. If during the second time step the arbitration process selects C, then the resulting message configuration is as in Figure 8.4(c). Message B is in a position such that it will be delayed by A, fulfilling the last condition of IV.B.2.b. Figure 8.4(d) shows the message locations after three time steps. All messages clear the node by the next time step.

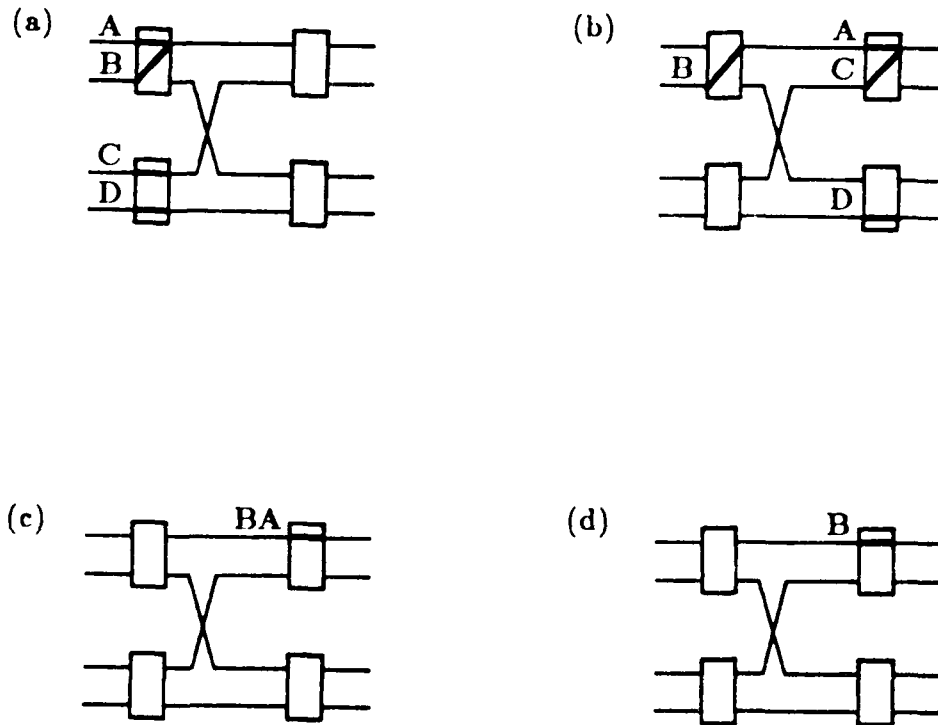


Figure 8.4 Message movement in a composite node for Table 8.1 entry IV.B.2.b. Messages are indicated by the letters A, B, C, and D. (a) Initial message position. A and B conflict in level 1. (b) Position after one time step. A and C conflict in level 2. (c) Position after two time steps. B experiences arbitration delay. (d) Position after three time steps.

As another example, Figure 8.5 illustrates entry IV.B.2.c of Table 8.1. The initial situation was arbitrarily chosen to be identical to that in Figure 8.4(a). Without loss of generality, A is arbitrarily selected to resolve the conflict in level 1, as in the previous example. If message A is selected to resolve the first conflict in level 2, shown in Figure 8.5(b), the result is as in Figure 8.5(c). The conditions of IV.B.2.c do not affect D. The second conflict in level 2, which distinguishes IV.B.2.c from IV.B.2.b, is shown in Figure 8.5(c). Figure 8.5(d) shows the result if C is selected to transit the box first.

If a single message is present at the inputs of the composite node it will transit in two time steps. The time is two because there are two levels. Thus,

$$P(t=2 | m=1) = 1.$$

Therefore,

$$E(t | m=1) = 2.$$

The following facts and notation are established for convenience in the analysis for $m = 2, 3,$ and 4 . There are $\binom{4}{2}$ ways two messages can be arranged at the inputs of a 4×4 switching element, and two of these involve having two messages at the inputs of a level 1 box for the composite switch. Thus,

$$P(\text{a level 1 box has 2 messages} | m=2) = \frac{2}{\binom{4}{2}} = \frac{1}{3}.$$

It is clear that

$$P(\text{a level 1 box has 2 messages} | m=3) = 1$$

and

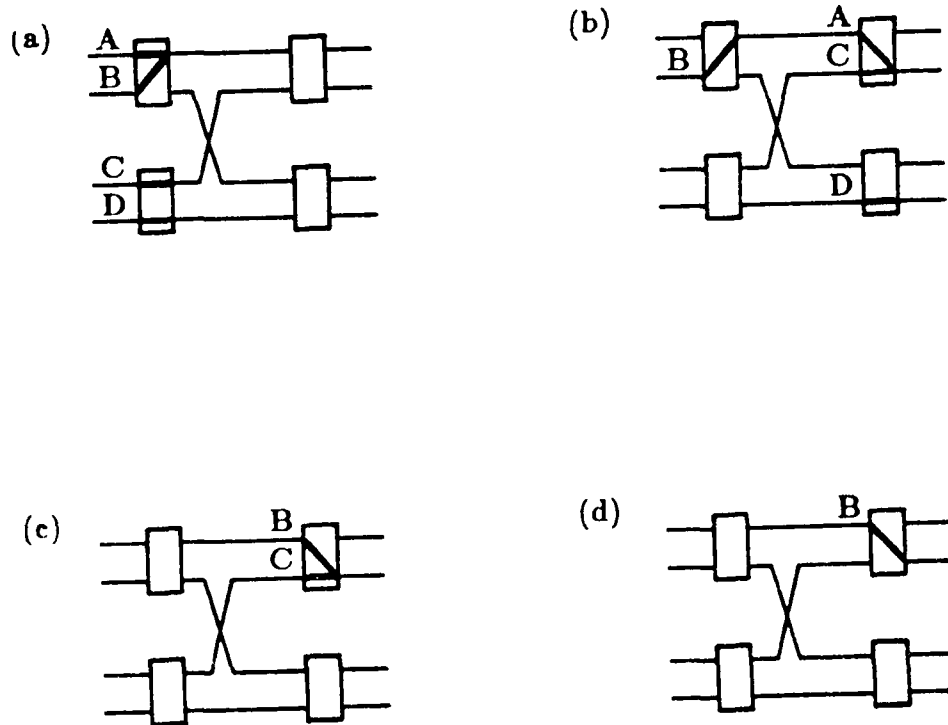


Figure 8.5 Message movement in a composite node for Table 8.1 entry IV.B.2.c. Messages are indicated by the letters A, B, C, and D. (a) Initial message position. A and B conflict in level 1. (b) Position after one time step. A and C conflict in level 2. (c) Position after two time steps. B and C subsequently conflict in level 2. (d) Position after three time steps.

$$P(\text{a level 1 box has 2 messages} \mid m=4) = 1.$$

Because two of four possible interchange box connections which may be needed by the messages involve conflict, and since destinations are random and uniformly distributed,

$$\begin{aligned} P(\text{conflict in an interchange box}) &= P(\text{no conflict in an interchange box}) \\ &= \frac{1}{2}. \end{aligned}$$

The following notation is used in the ensuing equations, where $i = 1$ or 2 :

$$\begin{aligned} 1. \quad P(iU) &= P(\text{no conflict in level } i \text{ upper box} \mid 2 \text{ messages at } \square \text{ inputs}) \\ &= \frac{1}{2}; \end{aligned}$$

$$\begin{aligned} 2. \quad P(iL) &= P(\text{no conflict in level } i \text{ lower box} \mid 2 \text{ messages at } \square \text{ inputs}) \\ &= \frac{1}{2}; \end{aligned}$$

$$3. \quad P(iX) = \frac{1}{2}, \text{ where } X = U \text{ or } L; \text{ and}$$

$$4. \quad P(i) = P(\text{no conflict in level } i \mid m=4) = P(iU) * P(iL) = \frac{1}{4}.$$

8.5.2 Passing Two Messages

For $m = 2$, transit times of two or three are possible. A time of two occurs only if there is no conflict. This corresponds to entry II.A.1 of Table 8.1. There are three cases to consider. First, if no box receives two messages

simultaneously then conflict is impossible, so

$$\begin{aligned}
 P(t=2, \text{ case 1} | m=2) &= P(\text{no box in any level receives 2 messages} | m=2) \\
 &= P(\text{no level 1 box has 2 messages} | m=2) * \\
 &\quad P(\text{no level 2 box has 2 messages} | m=2, \\
 &\quad \text{no level 1 box has 2 messages}) \\
 &= \left(1 - \frac{1}{3}\right) * \frac{1}{2} = \frac{1}{3}.
 \end{aligned}$$

Note that

$$P(\text{no level 2 box has 2 messages} | m=2, \text{ no level 1 box has 2 messages}) = \frac{1}{2}$$

because there are four ways two non-distinct messages can arrive at the inputs of level 2 boxes, and only two of the ways involve two messages at the inputs of a level 2 box. (See Figure 8.1 for aid in visualizing this.) Next, allow a level 1 box to receive two messages which do not conflict. There can be no conflict in level 2 in this case since no level 2 box will have two messages. So,

$$\begin{aligned}
 P(t=2, \text{ case 2} | m=2) &= P(\text{a level 1 box has 2 messages but no conflict} | m=2) \\
 &= P(\text{a level 1 box has 2 messages} | m=2) * P(1X) \\
 &= \frac{1}{3} * \frac{1}{2} = \frac{1}{6}.
 \end{aligned}$$

Finally, let a level 2 box receive two non-conflicting messages. Then,

$$\begin{aligned}
 P(t=2, \text{ case 3} | m=2) &= P(\text{a level 2 box has 2 messages but no conflict} | m=2) \\
 &= P(\text{no level 1 box has 2 messages} | m=2) * P(2X) * \\
 &\quad P(\text{a level 2 box has 2 messages} | m=2, \\
 &\quad \text{no level 1 box has 2 messages}) \\
 &= \left(1 - \frac{1}{3}\right) * \frac{1}{2} * \frac{1}{2} = \frac{1}{6}.
 \end{aligned}$$

Note that

$$\begin{aligned}
 P(\text{a level 2 box has 2 messages} | m=2, \text{ no level 1 box has 2 messages}) &= \\
 1 - P(\text{no level 2 box has 2 messages} | m=2, \text{ no level 1 box has 2 messages}) &= \\
 = \frac{1}{2}.
 \end{aligned}$$

With $m=2$ it is impossible to have both a level 1 and level 2 box receive two messages simultaneously. Thus, these three cases cover all possible ways to have no conflict with $m=2$. Summing the probabilities of the cases yields the desired result. Thus,

$$P(t=2 | m=2) = \frac{1}{3} + \frac{1}{6} + \frac{1}{6} = \frac{2}{3}.$$

A transit time of three (entries II.A.2 and II.B of Table 8.1) occurs if there is conflict. With $m=2$ conflict can occur in either level 1 or level 2, but not

both. So,

$$\begin{aligned}
 P(t=3 | m=2) &= P(\text{conflict} | m=2) \\
 &= P(\text{conflict in level 1} | m=2) + P(\text{conflict in level 2} | m=2) \\
 &= [1-P(1X)] * P(\text{a level 1 box has 2 messages} | m=2) + [1-P(2X)] * \\
 &\quad P(\text{a level 2 box has 2 messages} | m=2, \text{ no conflict in level 1}) \\
 &= \left(1 - \frac{1}{2}\right) * \frac{1}{3} + \left(1 - \frac{1}{2}\right) * \frac{1}{3} = \frac{1}{3}.
 \end{aligned}$$

Therefore,

$$E(t | m=2) = \sum_{i=2}^3 i * P(t=i | m=2) = 2 \frac{1}{3},$$

and

$$\sum_{i=2}^3 P(t=i | m=2) = \frac{2}{3} + \frac{1}{3} = 1,$$

as required.

8.5.3 Passing Three Messages

For $m=3$, transit times ranging from two to four are possible. Clearly, to achieve a time of two no conflict may occur (Table 8.1 entry III.A.1). So,

$$\begin{aligned}
 P(t=2 | m=3) &= P(\text{no conflict} | m=3) \\
 &= P(1X) * P(\text{a level 1 box has 2 messages} | m=3) * P(2X) * \\
 &\quad P(\text{a level 2 box has 2 messages} | m=3, \text{ no conflict in level 1})
 \end{aligned}$$

$$= \frac{1}{2} * 1 * \frac{1}{2} * 1 = \frac{1}{4} .$$

Note that

$$P(\text{a level 2 box has 2 messages} | m=3, \text{ no conflict in level 1}) = 1$$

because three messages must arrive simultaneously at level 2, given $m=3$ and no conflict occurred in level 1.

A time of three can occur in three ways for $m=3$. Assume first that conflict occurs only in level 1 (Table 8.1 entry III.B.1). Because first two messages reach level 2 simultaneously and then a single message,

$$\begin{aligned} P(t=3, \text{ case 1} | m=3) &= P(\text{conflict in level 1 only} | m=3) \\ &= P(\text{conflict in level 1} | m=3) * \\ &\quad P(\text{no conflict in level 2} | m=3, \text{ conflict in level 1}) \\ &= \frac{1}{2} * \frac{3}{4} = \frac{3}{8} . \end{aligned}$$

The term $P(\text{no conflict in level 2} | m=3, \text{ conflict in level 1})$ in the above equation must be used instead of $P(\text{no conflict in level 2} | m=3)$ because the level 1 outputs used by two messages arriving at level 2 have an effect on the probability of conflict in level 2. For example, if the two messages come from the same level 1 box (implies no conflict in level 1) then conflict in level 2 is impossible since the messages must use different level 2 boxes. For this case of the analysis conflict occurs in level 1, so the two messages first arriving in level 2 must come from distinct level 1 boxes. The probability these messages do conflict is

$$P(\text{they arrive at the same level 2 box}) * P(2X) = \frac{1}{2} * \frac{1}{2} = \frac{1}{4} .$$

So,

$$P(\text{no conflict in level 2} | m=3, \text{ level 1 conflict}) = 1 - \frac{1}{4} = \frac{3}{4} .$$

For conflict in level 2 only (Table 8.1 entry III.A.2),

$$P(t=3, \text{ case 2} | m=3) = P(\text{no conflict in level 1} | m=3) *$$

$$P(\text{conflict in level 2} | m=3)$$

$$= \frac{1}{2} * \frac{1}{2} = \frac{1}{4} .$$

Finally, a time of three occurs if there is a single conflict in level 1 and a single conflict in level 2 without any arbitration delay or subsequent conflicts (Table 8.1 entry III.B.2.a). Thus,

$$P(t=3, \text{ case 3} | m=3) = P(\text{conflict in level 1} | m=3) *$$

$$P(\text{conflict in level 2} | m=3, \text{ conflict in level 1}) *$$

$$\frac{1}{2} * P(2X)$$

$$= \frac{1}{2} * \left(1 - \frac{3}{4}\right) * \frac{1}{2} * \frac{1}{2} = \frac{1}{32} .$$

The first two factors of this equation represent the probability of having both a level 1 and a conflict in level 2 with $m=3$. The $\frac{1}{2}$ factor is the probability that the conflict in level 2 is resolved without resulting arbitration delay. The $P(2X)$ factor is the probability the last two messages to exit the node do not

conflict in level 2. Combining the above yields

$$P(t=3 | m=3) = \sum_{i=1}^3 P(t=3, \text{ case } i | m=3) = \frac{3}{8} + \frac{1}{4} + \frac{1}{32} = \frac{5}{8}$$

A time of four occurs when there is conflict at both levels with an arbitration delay or subsequent conflict (Table 8.1 entries III.B.2.b and III.B.2.c). Thus,

$$P(t=4 | m=3) = P(\text{conflict in levels 1 and 2} | m=3)$$

$$= P(\text{conflict in level 1} | m=3) *$$

$$P(\text{conflict in level 2} | m=3, \text{ conflict in level 1}) *$$

$$\left[\frac{1}{2} + \frac{1}{2} * P(2X) \right]$$

$$= \frac{1}{2} * \left(1 - \frac{3}{4} \right) * \left[\frac{1}{2} + \frac{1}{2} * \frac{1}{2} \right] = \frac{3}{32}$$

Note that $P(\text{conflict in level 2} | m=3, \text{ conflict in level 1}) = 1 - P(\text{no conflict in level 2} | m=3, \text{ conflict in level 1}) = 1 - \frac{3}{4}$, as appears in the equation for $P(t=3, \text{ case 1} | m=3)$.

In conclusion,

$$\sum_{i=2}^4 P(t=i | m=3) = \frac{1}{4} + \frac{5}{8} + \frac{1}{8} = 1$$

and

$$E(t | m=3) = \sum_{i=2}^4 i * P(t=i | m=3) = 2 \frac{27}{32} .$$

8.5.4 Passing Four Messages

Now consider the probabilities of different times to pass four messages through the composite node. Only two time steps are needed when there is no conflict (Table 8.1 entry IV.A.1). So,

$$P(t=2 | m=4) = P(1U) * P(1L) * P(2U) * P(2L) = \frac{1}{16} .$$

There are five cases to consider for a time of three. First assume no conflicts occur in level 1 (Table 8.1 entry IV.A.2). Then one or two conflicts must occur in level 2. Thus,

$$P(t=3, \text{ case 1} | m=4) = P(1) * (1 - P(2)) = \frac{3}{16} .$$

Next, assume exactly one level 1 interchange box has a conflict and there is no subsequent conflict in level 2 (Table 8.1 entry IV.B.1.a). Hence,

$$P(t=3, \text{ case 2} | m=4) = [(1 - P(1U)) * P(1L) + P(1U) * (1 - P(1L))] * P(2X) = \frac{1}{4} .$$

In the above expression, $P(2X)$ is used because one of the level 2 boxes must process three messages (one arriving later) while the other processes just one. The former must be conflict free, and the latter has no constraint.

Next, consider conflict at both levels (Table 8.1 entry IV.B.2.a). Assume the four messages are A, B, C, and D, and there is one conflict in level 1 for example, between A and B, so all but message B go to level 2 after one time step. Also assume there is one conflict in level 2, between A and C (or D).

First, depending upon which message, A or C, is chosen by the arbitration logic, B may find it must wait for A even if its output request differs from that of A, or for C if B and C do request the same output. Second, if A is chosen to transit level 2 before C, and B and C request different output lines, then B will not be delayed at level 2. The former situation requires four time steps, and the latter (Table 8.1 entry IV.B.2.a) requires three. Figure 8.6 illustrates one instance of the latter situation. The probability of the latter is

$$\begin{aligned}
 P(t=3, \text{ case 3} | m=4) &= [(1 - P(1U)) * P(1L) + P(1U) * (1 - P(1L))] * \\
 &\quad (1 - P(2X)) * \frac{1}{2} * P(2X) \\
 &= \frac{1}{16} .
 \end{aligned}$$

The first factor is the probability that exactly one level 1 box has a conflict. The next factor is the probability that the first message from the level 1 box which had a conflict, call this message M, also has a conflict in level 2. The $\frac{1}{2}$ is the probability that M will be chosen to transit level 2 first. The last factor is the probability that the two delayed messages (B and C in the example) do not conflict.

Now assume conflict in both level 1 boxes and that both level 2 boxes receive messages (this happens half the time there are two conflicts in level 1). Some of the instances of Table 8.1 entry IV.B.1.b fit this assumption. There is no possibility of conflict in level 2 boxes, so

$$P(t=3, \text{ case 4} | m=4) = \frac{1}{2} * (1 - P(1U)) * (1 - P(1L)) = \frac{1}{8} .$$

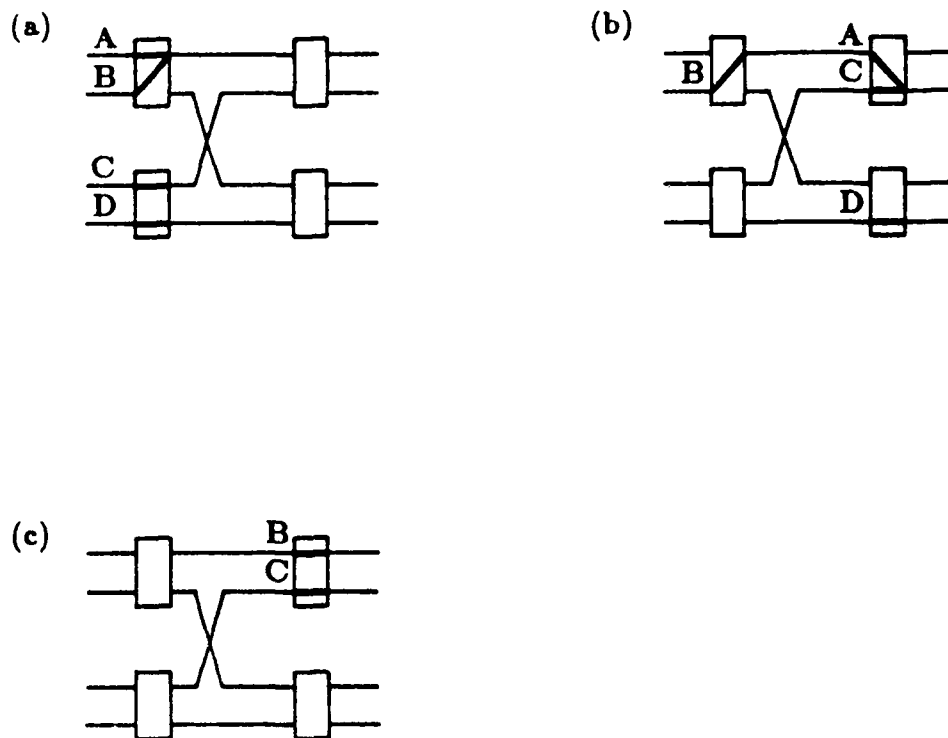


Figure 8.6 Message movement in a composite node corresponding to one element in the set of message flow patterns covered by $P(t=3, \text{ case } 3 \mid m=4)$. (a) Initial message position. A and B conflict in level 1. (b) Position after one time step. A and C conflict in level 2. (c) Position after two time steps. No remaining conflicts.

Finally, assume conflict in both level 1 boxes but only one level 2 box receives messages and there is no conflict for either message pair that transits that box. This describes the rest of the instances of Table 8.1 entry IV.B.1.b.

Then

$$P(t=3, \text{ case 5} | m=4) = \frac{1}{2} * (1 - P(1U)) * (1 - P(1L)) * P(2X) * P(2X) = \frac{1}{32}$$

The probability that all messages transit the composite node in three time steps is

$$\begin{aligned} P(t=3 | m=4) &= \sum_{i=1}^5 P(t=3, \text{ case } i | m=4) \\ &= \frac{3}{16} + \frac{1}{4} + \frac{1}{16} + \frac{1}{8} + \frac{1}{32} = \frac{21}{32} \end{aligned}$$

For a time of four, there are four cases. One involves the situation of the third case for a time of three. There are two ways to obtain a time of four in that situation: (1) message B enters a non-empty queue in level 2 (Table 8.1 entry IV.B.2.b (see Figure 8.4)) and (2) B enters an empty queue but conflicts with the remaining message (C or D) (Table 8.1 entry IV.B.2.c (see Figure 8.5)). Thus,

$$\begin{aligned} P(t=4, \text{ case 1} | m=4) &= [(1 - P(1U)) * P(1L) + P(1U) * (1 - P(1L))] * \\ &\quad \left[\frac{1}{2} * (1 - P(2X)) + \frac{1}{2} * (1 - P(2X)) * (1 - P(2X)) \right] \\ &= \frac{3}{16} \end{aligned}$$

The first factor is the probability of conflict in exactly one level 1 box. The first term of the second factor is the probability B enters a non-empty queue.

This term has two factors: $(1 - P(2X))$ is the probability of the conflict in level 2 necessary to force a message to remain at the input of the level 2 box B will use until B arrives, and $\frac{1}{2}$ is the probability that the message is at the level 2 box input B will use. The second term of the second factor is the probability that B enters an empty level 2 box input but conflicts with the other message at that box. Again, $(1 - P(2X))$ is the probability of the first conflict in level 2. The $\frac{1}{2}$ is the probability B arrives at the empty level 2 box input. The second $(1 - P(2X))$ is the probability B conflicts with the remaining message (C or D).

Now assume conflict in both level 1 boxes and that only one level 2 box receives messages (this happens half the time there are two conflicts in level 1). There are then three ways (cases 2, 3, and 4) a time of four can occur. In case 2, the first two messages reaching the box in level 2 conflict, but there are no subsequent conflicts (Table 8.1 entry IV.B.3.b), so

$$P(t=4, \text{ case 2} | m=4) = \frac{1}{2} * (1 - P(1U)) * (1 - P(1L)) * (1 - P(2X)) * P(2X) = \frac{1}{32} .$$

The $\frac{1}{2}$ factor is the probability that only one level 2 box receives messages given there are conflicts in both level 1 boxes, denoted by the factors $(1 - P(1U))$ and $(1 - P(1L))$. The probability that the first two messages to reach level 2 conflict, but there are no subsequent conflicts, is $(1 - P(2X)) * P(2X)$.

In case 3, the first pair of messages arriving at level 2 do not conflict, but the second pair do (Table 8.1 entry IV.B.3.a), yielding

$$P(t=4, \text{ case 3} | m=4) = \frac{1}{2} * (1 - P(1U)) * (1 - P(1L)) * P(2X) * (1 - P(2X)) = \frac{1}{32} .$$

This equation is derived in a similar way to that for $P(t=4, \text{ case 2} | m=4)$.

Figure 8.7 is provided as an aid to understanding the message movement corresponding to case 4. In this case, the first pair of messages arriving at level 2, A and C, conflict. A, for example, transits and C remains while B and D arrive on distinct inputs. D is behind C in the queue. Assume there is a second conflict between B and C. Depending upon which (B or C) is chosen by the priority logic (either is equally likely), either (1) B and D are at the front of the queues or (2) C and D are in the same queue. To obtain a time of four, B and D must be at the front of the queues and not conflict (Table 8.1 entry IV.B.3.c), otherwise a time of five results. Hence,

$$\begin{aligned}
 P(t=4, \text{ case 4} | m=4) &= \frac{1}{2} * (1-P(1U)) * (1-P(1L)) * (1-P(2X)) * \\
 &\quad (1-P(2X)) * \frac{1}{2} * P(2X) \\
 &= \frac{1}{128} .
 \end{aligned}$$

The first four factors of the equation appear for the same reason as the first four factors in the expression for $P(t=4, \text{ case 2} | m=4)$. The fifth factor, $(1-P(2X))$, is the probability that B and C conflict as assumed. The $\frac{1}{2}$ is the probability that C is chosen to resolve the conflict so that B and D are ready to transit level 2 on the next time step. They will do so if they do not conflict. This has a probability of $P(2X)$, the last factor. The probability of a time of four is

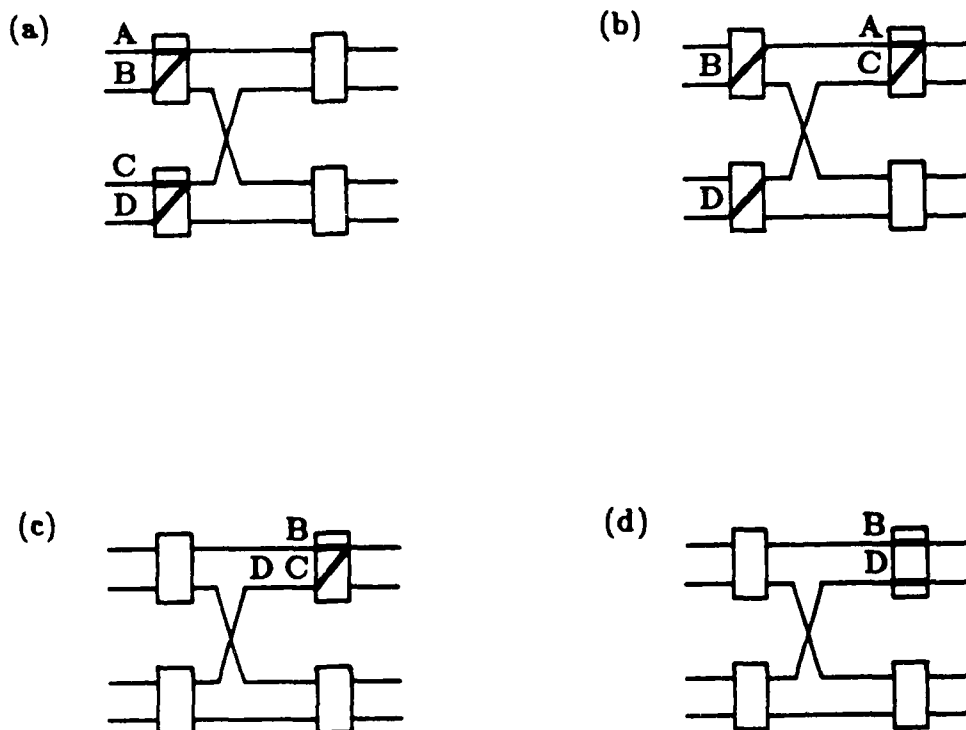


Figure 8.7 Message movement in a composite node corresponding to one element in the set of message flow patterns covered by $P(t=4, \text{case } 4 \mid m=4)$. (a) Initial message position. (b) Position after one time step. (c) Position after two time steps. (d) Position after three time steps.

$$\begin{aligned}
 P(t=4 | m=4) &= \sum_{i=1}^4 P(t=4, \text{ case } i | m=4) \\
 &= \frac{3}{16} + \frac{1}{32} + \frac{1}{32} + \frac{1}{128} = \frac{33}{128} .
 \end{aligned}$$

Finally, there is only one way a time of five can occur and that begins with the situation of case four above, where a time of four or five can occur. When B and D are at the front of the queues and conflict, or C and D are in the same queue (Table 8.1 entries IV.B.3.d and IV.B.3.e), $t = 5$. So,

$$\begin{aligned}
 P(t=5 | m=4) &= \frac{1}{2} * (1 - P(1U)) * (1 - P(1L)) * (1 - P(2X)) * \\
 &\quad \left[(1 - P(2X)) * \frac{1}{2} * (1 - P(2X)) + (1 - P(2X)) * \frac{1}{2} \right] \\
 &= \frac{3}{128} .
 \end{aligned}$$

The first four factors of this equation appear for the same reason as the first four factors in the expressions for $P(t=4, \text{ case } 2 | m=4)$ and $P(t=4, \text{ case } 4 | m=4)$. The last factor consists of two terms. The first is the probability B and C conflict and C was chosen to resolve the conflict, setting the stage for B and D to use a level 2 box simultaneously and conflict. The second term is the probability B and C conflict, but C was not chosen, forcing D to follow C through the same level 2 box input.

Therefore,

$$E(t | m=4) = \sum_{i=2}^5 i * P(t=i | m=4) = 3 \frac{31}{128} .$$

and

$$\sum_{i=2}^5 P(t=i | m=4) = \frac{1}{16} + \frac{21}{32} + \frac{33}{128} + \frac{3}{128} = 1.$$

8.5.5 Permutation Delay Probability

As in Section 8.4, the preceding analysis is most applicable to MIMD environments. An analysis for the composite node in SIMD mode, analogous to that given for the crossbar node, can be performed. The composite node can perform $2^4 = 16$ permutations. Thus,

$$P(\text{delay} | m=4) = 1 - P(\text{no conflict} | m = 4) = 1 - \frac{2^4}{4^4} = 0.938.$$

Thus, 93.8 percent of the time four messages try to pass through a composite node simultaneously there will be conflict, assuming random node output selection with uniform distribution. Again, $P(\text{delay} | m = 4)$ is likely to be of limited use for understanding network level, as opposed to switching element level, behavior.

8.5.6 Summary

The results of the analysis for the two nodes are summarized in Table 8.2. An obvious difference between the two implementations is that twice as much time is required to transit the composite node as the crossbar node when there is no contention. A more subtle difference in their blocking can be observed by comparing the incremental difference in expected delay that each exhibits as m is varied. This is the discrete derivative of delay and gives an indication of the degree of conflict as a function of m .

Table 8.2 Summary of transit time performance and incremental delay characteristics for the crossbar and composite nodes.

Number of Messages (m)	Crossbar		Composite	
	Expected Transit Time $E(t m)$	Incremental Difference	Expected Transit Time $E(t m)$	Incremental Difference
1	1	-	2	-
2	1.25	0.25	2.33	0.33
3	1.69	0.44	2.84	0.51
4	2.13	0.44	3.24	0.36
Total Difference	-	1.13	-	1.24

In the crossbar node the incremental difference in delay is a nondecreasing sequence with increasing m . For the composite node there is a noticeable "hump." Increasing m from two to three produces a greater increase in $E(t|m)$ than an increase from three to four. This can be explained in the following way. When $m = 2$ there is a two-thirds probability that the two messages enter different level 1 interchange boxes, since there are $\binom{4}{2} = 6$ ways two messages can occupy the four level 1 inputs and four of these ways do not involve two messages at a single level 1 box. This reduces the chance for conflict in level 1 in the composite node. However, when $m = 3$ one level 1 box must have two messages to process, increasing the chance for conflict in the node significantly. A smaller increase occurs for $m = 4$, because it is then possible for two conflicts to occur in the same level.

Node level behavior can provide insight into overall network behavior. In a multistage cube-type network, the average delay should increase more rapidly as a message loading threshold is reached and then taper off as full loading is reached. On the other hand, a complete crossbar network of the same size should exhibit a nondecreasing sequence of incremental increases in delay. These predictions should of course be verified by simulation.

Finally, it can be observed that the total increase in delay is more for the composite node. This difference is due to the fact that the composite node is a blocking network, while the crossbar is not. The composite node introduces conflict above and beyond that due to coincident message destinations.

8.6 Network Implementation and Performance

An ESC network can be constructed from 4×4 switching nodes using the following method. Needed are $(n-1)/2$ stages of $N/4$ 4×4 switching nodes each, for n odd. The input and output stages must be implemented with interchange boxes as usual. In the ESC topology, at stage i , $0 \leq i < n$, the two inputs to an interchange box have addresses that differ only in the i^{th} bit position. Let n be odd (a way of using 4×4 switching nodes with n even will be described later). Either append 0 to the right of the least significant bit of the binary representation of all labels, or append 1. This allows the appropriate bits of the binary representation to be paired during conversion to base four notation. Convert all link labels to base four representation. For purposes of the construction method description, let the $(n-1)/2$ stages of $N/4$ 4×4 switches be numbered $\left\lfloor \frac{n-1}{2} \right\rfloor - 1, \dots, 1, 0$ (from input to output). At stage i , $0 \leq i < (n-1)/2$, the four inputs to a 4×4 switching node must have labels that differ only in the i^{th} position of their base four representation. The link with a 0 in the i^{th} position of its address connects to the top input of the 4×4 node, 2 to the next input, 1 to the next input, and finally 3 to the bottom input [Smi81]. The outputs of the 4×4 switching nodes have the same labels as the input lines, but in increasing order, i.e., the top output label has a 0 in the i^{th} position, next 1, next 2, and the bottom 3. Figure 8.8 shows a composite node labeled as described. The bits of the corresponding binary labeling for the constituent boxes are shown internal to the node. When composite nodes are used, making connections in the above manner creates an ESC network. When crossbars nodes are used, a network is created whose capabilities are a superset of those of the ESC network. Figure 8.9 shows how

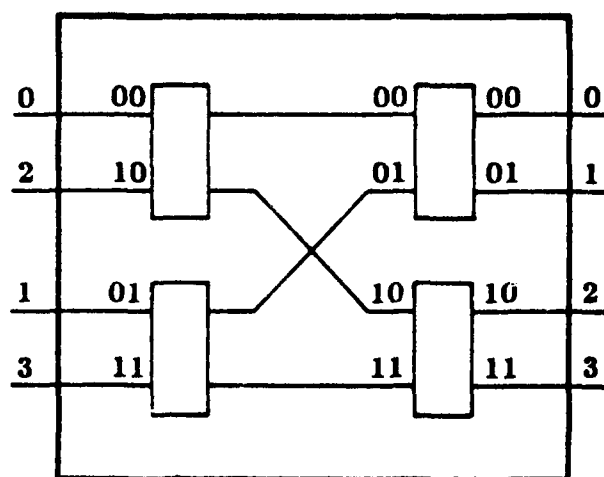


Figure 8.8 A composite node labeled for use in construction of an ESC network.

4×4 switches can be used in the construction of an ESC, or ESC-like, (if crossbar switches are used) network with $N = 8$.

The ESC network provides for individual box control. Because there are $N(n + 1)/2$ interchange boxes in an N -input network, but stage n is normally bypassed, there are $2^{Nn/2}$ distinct network settings. Each of these settings corresponds to a unique permutation of network inputs to outputs. Since an ESC network built from composite nodes results in a network identical to one built from interchange boxes, the number of permutations performable using either implementation is the same.

If crossbar nodes are used, the number of performable permutations is greater than with the composite nodes. This is because each crossbar node has $4! = 24$ settings which are permutations. As with the interchange box implementation, the network performs a permutation of inputs to outputs if and only if each crossbar node is set to a one-to-one connection of inputs to outputs. A network with N such that n is odd can be constructed from 4×4 crossbar nodes, except for the input and output stages. Thus, $\left\lfloor \frac{n-1}{2} \right\rfloor * \left\lfloor \frac{N}{4} \right\rfloor$ 4×4 crossbar nodes are used. Such a network, with stage n bypassed, can perform $2^{N/2}(4!)^{N(n-1)/8}$ permutations. These permutations are unique since there is only one path from a given input to a given output. If one switching node setting is changed, there is no other switching node whose setting can be changed to yield the original permutation. Table 8.3 compares the number of permutations performable with composite and crossbar nodes.

When N is such that n is even, then one way the network can be implemented is to use 4×4 crossbar or composite nodes for all but three stages. Any one stage i in the original ESC network, where i is odd, can be

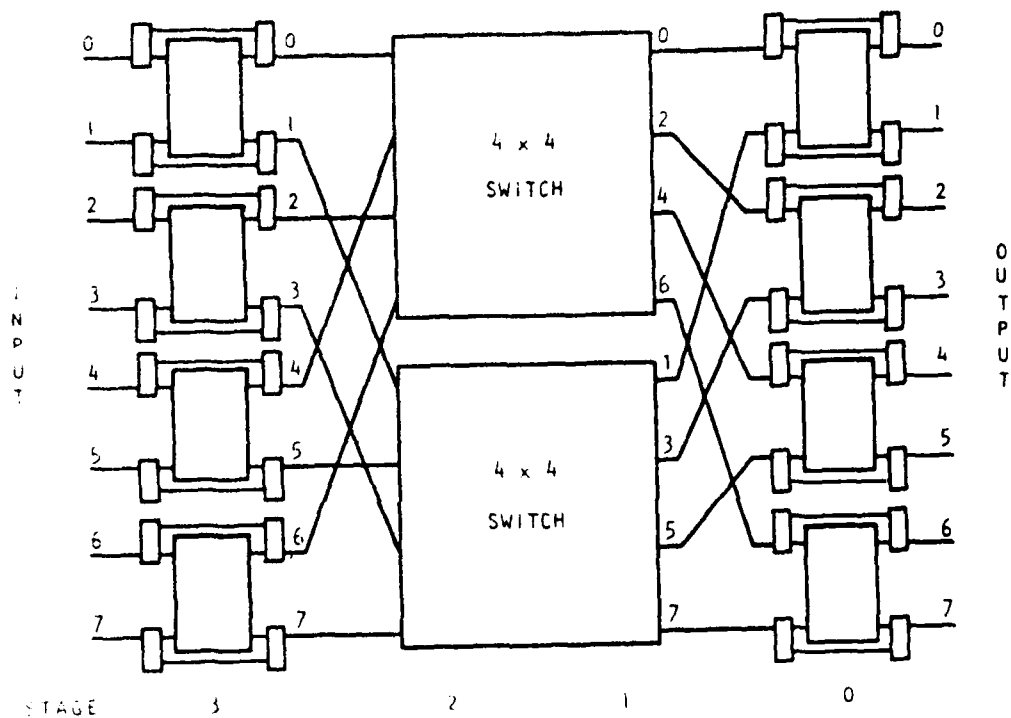


Figure 8.9 Construction of an ESC with $N = 8$ and 4×4 switching elements.

Table 8.3 Comparison of permuting capabilities of ESC networks implemented with the maximum number of composite and crossbar nodes (stage n bypassed) for various values of N.

N	Number of Permutations	
	Composite	Crossbar
4	16	16
8	4096	9216
16	4.29×10^9	2.17×10^{10}
32	1.21×10^{24}	7.94×10^{26}
64	6.28×10^{57}	2.71×10^{63}
128	7.27×10^{134}	5.84×10^{151}
256	1.80×10^{308}	1.16×10^{342}
512	3.74×10^{693}	5.39×10^{783}
1024	1.88×10^{1541}	3.90×10^{1721}

constructed with interchange boxes or with 4×4 crossbar nodes which are limited to performing only a single cube function. The link labels should be converted to base four to specify the 4×4 node connections after a 0 (or 1) has been inserted immediately to the right of bit position i in the binary representation of the link labels. This allows the appropriate bits to be paired during conversion to base four notation. Again for construction purposes, the stages of 4×4 switches can this time be numbered $\left\lfloor \frac{n}{2} \right\rfloor - 1, \dots, 1, 0$. Stage $\frac{i-1}{2}$ in this new numbering scheme is the stage implemented with interchange boxes or limited 4×4 crossbar nodes. For n even and with stage n bypassed, the network can perform $2^N(4!)^{N(n-2)/8}$ unique permutations if crossbar nodes are used, and $2^{Nn/2}$ with composite nodes, as before.

8.7 Conclusions

The crossbar node is always faster at passing messages than the composite node. If the connection requests do not conflict in the composite node, the crossbar is twice as fast. When the connection requests of the messages at a given switching element form a permutation that a composite node cannot pass without conflict, but that a crossbar can, it takes three times longer for all messages to exit the composite node. Finally, as the number of messages increases the degree of conflict in the composite node rises more rapidly than in the crossbar node.

The performance of basic building blocks for cube-type multistage interconnection networks has been examined. The 4-input/4-output crossbar node has been compared to a 4-input/4-output composite node constructed from 2-input/2-output interchange boxes. Capabilities of these two switching

element design approaches were quantified. Implementation of an ESC, or ESC-like, network using 4-input/4-output switches was discussed. The results presented give designers of parallel computer systems systems additional information to apply to interconnection network design.

CHAPTER 9

STUDY OF IMAGE CONTOUR EXTRACTION WITH NETWORK AND SYSTEM IMPLICATIONS

9.1 Introduction

Digital image processing has long been recognized as well-suited for parallel processing. Many individual image processing algorithms and their formulations as parallel algorithms have been studied, including, for example, image coding [MDS82], correlation [Ack77, SSF82], histogramming [SSK81], line segment generation [Sta74], resampling [WaS82], segmentation [Dou82], and two-dimensional FFT [MSS80b]. Each part of this body of work illuminates some aspect of image processing with a parallel processing system.

The whole can sometimes be greater than the sum of its parts. Similarly, study of an image processing scenario, a task consisting of multiple, basic image processing algorithms such as those above, can yield insight not forthcoming from consideration of the elements individually. A succession of processing steps is more characteristic of actual machine use than the load associated with any single algorithm. The interaction between image processing steps may well influence the scenario-level processing method of choice.

Portions of this chapter were developed with David L. Tuomenoksa, James T. Kuehn, O. Robert Mitchell, and Kirk A. Dunkelberger

As stated in Chapter 2, Section 2.2, PASM is intended to be suited for several applications, including image processing, and will be designed to meet that goal. The work in this chapter contributes to that effort. The nature of the anticipated task load is a significant consideration in the selection of an interconnection network for a parallel processor as well. The capabilities of the ESC and its place among other fault-tolerant multistage interconnection networks have been established. The next logical step in a study of the ESC is to ascertain its utility for performing the communication role in a multiprocessor computer system, in particular, PASM.

Contour extraction is the focus of this chapter; it was selected for two reasons. First, contour extraction is a key image processing step in applications ranging from computer-assisted cartography to industrial inspection [ChH82, Jar80]. Digital image processing will continue to increase in importance for these applications. Second, contour extraction presents a multifaceted challenge to a parallel computer; hence, it illuminates a number of issues in parallel processor design [TAS83].

9.2 Contour Extraction

Contour extraction consists of two operations: segmentation and contour tracing. Segmentation divides an image into various regions; the goal is for these regions to correspond closely with the depicted background and with objects of interest. Contour tracing identifies the boundaries of regions. Object boundaries are the items interest.

There are many ways to accomplish segmentation [RiA77]. One uses thresholding, a technique by which an image is mapped onto two brightness, or *gray* levels. Typically, all portions of an image at or below a brightness

threshold, or level, are set to black, and all above are set to white. Determining a single threshold that yields a useful segmentation for a given image can be difficult; this is particularly true when the average brightness level varies significantly among various areas in the image. It may be that no satisfactory single threshold exists for a given image. Thresholding can be generalized to using any well defined image feature, and not just pixel gray levels.

If an image is divided into sufficiently small subimages, the lighting level will be approximately uniform within a given subimage despite significant image-wide variation. Thresholding independently within each subimage can succeed in capturing objects of interest throughout the image when a single, global threshold will fail.

Contour tracing algorithms depend upon the topological properties of adjacency and connectedness for contours. Let (x,y) be the coordinates of a pixel. The four horizontal and vertical neighbors of (x,y) , namely pixels at $(x-1,y)$, $(x,y-1)$, $(x,y+1)$, and $(x+1,y)$ are the *4-neighbors* of (x,y) and are *4-adjacent* to (x,y) [RoK82]. Similarly, including the diagonally adjacent neighbor pixels at $(x-1,y-1)$, $(x-1,y+1)$, $(x+1,y-1)$, and $(x+1,y+1)$ with the 4-neighbors of (x,y) yields the *8-neighbors* of (x,y) , which are *8-adjacent* to (x,y) . The pixels of objects of interest are assumed to be 8-adjacent; this is consistent with intuition about the connectedness of objects. Hence, contour pixels, being the border pixels of an object, are 8-adjacent. Consequently, the background is 4-adjacent. Consider the pattern of pixels in Figure 9.1. If the pixels labeled with 1s represent object pixels and those with 0s represent background pixels, then the two object pixels are connected and the two background pixels are not.

1 0
0 1

Figure 9.1 Two 8-adjacent 1s and two non-4-adjacent 0s.

Contour extraction can be preceded by processing such as radiometric correction and rectification. Subsequent processing, once contours have been extracted, may range from simple contour highlighting to shape analysis and classification involving significant additional computation. The specific context of the contour extraction scenario would depend on the application. Processing of an image subsequent to contour extraction is beyond the scope of this chapter.

9.3 Overlapped Subimage Method

A set of serial algorithms that produces interpretation results from digitized imagery using the methods outlined in the preceding section has been implemented at Purdue University on a VAX-11/780 computer. This method analyzes an image in square subimages that are processed independently and sequentially. Objects must be completely contained within some subimage if they are to be found. To facilitate this, subimages are overlapped 50 percent horizontally and vertically. The approach is referred to as the *overlapped subimage method* for this reason. The overlap guarantees that objects restricted to a maximum dimension of $(r/2) - 1$ pixels (picture elements), for a subimage size of $r \times r$, will be contained in their entirety in some subimage. A typical value for r is 256; a typical value for image size is 5120×5120 .

Edge information has been used to guide threshold selection to achieve segmentation that more accurately separates objects of interest from other parts of an image [Mil79, NeP82]. The first step of the overlapped subimage method is to choose thresholds using *edge-guided thresholding (EGT)* [SuR82] to segment the image. It selects threshold levels using an edge-matching criterion instead of the classical technique using local minimum values in the

image gray level histogram [PrM66]. Initially, an *edge image* is generated using the Sobel edge operator [DuH73]. A figure of merit, indicating how well a given thresholded version of the input image matches edges in the edge image, is then computed. Thresholds with high figures of merit are selected for requantizing the input image prior to contour tracing. A median filter [GaW81] can be applied immediately after thresholding to remove isolated noise artifacts. Frequently, EGT gives better results than the histogram method because it is able to detect small objects not discernible from the histogram [SuR82]. The EGT algorithm will be detailed in the next section.

Because objects are contained within some subimage (by definition) when using the overlapped approach, a simple procedure suffices for tracing the contours generated by segmentation. First, only complete, or *closed* contours need be traced. Second, if a subimage is scanned left to right and from top to bottom then objects will be first encountered at the leftmost pixel of their uppermost row of pixels. Thus, the contour tracing algorithm can be formulated to take advantage of the fact that the location of the region bounded by the contour to be traced is known: namely, all region pixels are either to the right, below, or both right and below the first found pixel. Contours are stored as a sequence of x-y coordinates.

The overlapped subimage approach to contour extraction was described in an image shape analysis method [MRF81] directed toward classifying small well-defined objects such as buildings and airplanes. In [MRF81] extracted contours are used for object classification by comparing the contours with prototypic object models using either Fourier descriptors [WaW80] or standard moments [Hu62, Tea80]. The overlapped subimage method yields good results for this application, despite large brightness variations across an image, but is

computationally intensive. Execution time can be significantly reduced by exploiting the parallelism inherent in contour extraction.

9.4 Non-Overlapped Subimage Method

The overlapped subimage method as it stands is not particularly well suited for parallel processing. If each overlapped subimage is assigned to a processor then either much image data must be passed between processors or what amounts to four copies of the image must be stored. This is because the overlapping causes any one pixel (except for a few near the corners of the image) to appear in four subimages. For parallel processing a non-overlapped approach will be used. With this approach each PE will be assigned one subimage, and subimages will not overlap. Note that assigning more than one subimage to a PE using the non-overlapped method is logically equivalent to assigning only one subimage, but with a larger size.

An $R \times R$ pixel image is represented by an array of R^2 pixels, where the value of each pixel is assumed to be an unsigned integer representing one of the possible gray levels. To implement contour extraction on an SIMD/MIMD machine of N PEs, assume that the PEs are logically configured as a $\sqrt{N} \times \sqrt{N}$ grid, on which the $R \times R$ image is superimposed, i.e., each processor has an $(R/\sqrt{N}) \times (R/\sqrt{N})$ subimage (see Figure 9.2(a)). For example, if $R = 5120$, each PE stores a 160×160 subimage. Each pixel is uniquely addressed by its i - x - y coordinates, where x and y are the x - y coordinates of the pixel in the subimage contained in PE_i . This data allocation minimizes the perimeter of a subimage; the value of this is discussed in the next section.

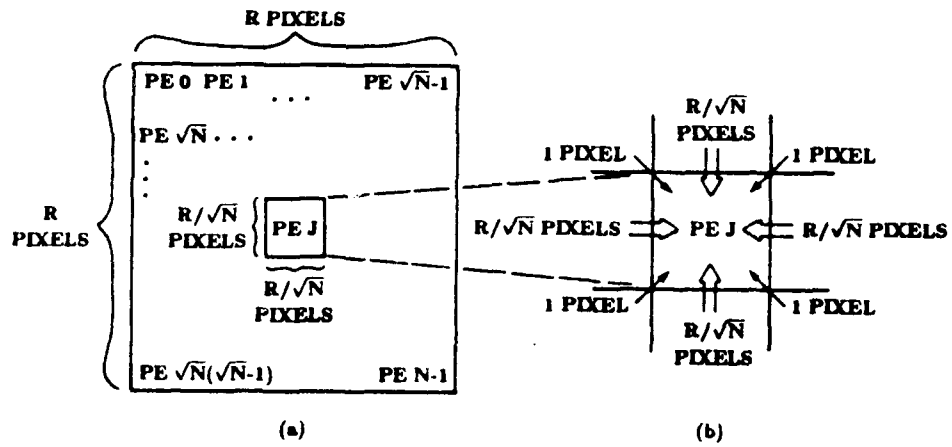


Figure 9.2 (a) Data allocation for a $R \times R$ image using N PEs.
 (b) Data transfers needed for Sobel edge operator.

AD-R167 621

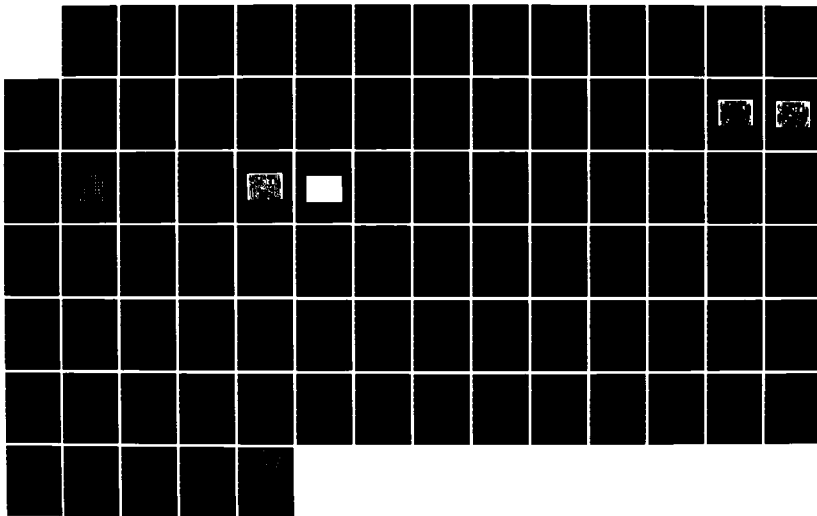
DISTRIBUTED COMPUTING FOR SIGNAL PROCESSING: MODELING
OF ASYNCHRONOUS PAR. (U) PURDUE UNIV LAFAYETTE IN
8 8 ROWMS DEC 84 AR0-18790.17-EL-APP-C DRAG23-82-K-0181

4/4

UNCLASSIFIED

F/G 9/2

ML





1.0



2.2



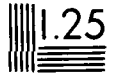
1.1



2.0



1.8



1.25



1.4



1.6

Resolution Test Chart

9.4.1 EGT Algorithm

The parallel EGT algorithm operates on each subimage independently and is performed by all PEs simultaneously. It consists of three major conceptual steps. First, an edge image is generated, then a figure of merit is computed for every possible threshold, and finally, threshold levels are selected corresponding to local maxima in the figure of merit.

Let the image I be $R \times R$ and $I(x,y)$ be a pixel, where $0 \leq x,y \leq R-1$. Let the subimage assigned to a processor be $r \times r$, where $r = R/\sqrt{N}$. In the first step, the Sobel operator is performed by each PE for its subimage, generating the edge image. Data from adjacent subimages is acquired via the interconnection network, all subimages receiving pixels simultaneously. The number of parallel transfers needed for an $R \times R$ image is $4 * (R/\sqrt{N} + 1)$, as shown in Figure 9.2(b). The Sobel operator (ignoring image border pixels for clarity) is given in Figure 9.3. The value $g(x,y)$ represents the gradient at pixel (x,y) ; these values form the edge image. High edge-image pixel values indicate the presence of an edge. The pixels immediately adjacent to the edges of a subimage are needed in the Sobel calculation for those pixels at the edges of a subimage. Thus, minimizing the perimeter of a subimage also minimizes the number of pixels that must either be acquired from the adjacent subimages or initially loaded into the memory of more than one PE. For this reason square subimages are preferable to rectangular ones. Typically, the Sobel value for pixels at the edge of an image are either not calculated or are calculated assuming that pixel locations falling outside the image correspond to background pixels. In this work pixel locations outside the image are assumed to have the value of background pixels.

```

for x = 0 to r-1 do begin
  for y = 0 to r-1 do begin

```

$$sx(x,y) = \frac{1}{4} \left[(I(x-1,y-1) + 2*I(x-1,y) + I(x-1,y+1)) \right. \\ \left. - (I(x+1,y-1) + 2*I(x+1,y) + I(x+1,y+1)) \right]$$

$$sy(x,y) = \frac{1}{4} \left[(I(x-1,y-1) + 2*I(x,y-1) + I(x+1,y-1)) \right. \\ \left. - (I(x-1,y+1) + 2*I(x,y+1) + I(x+1,y+1)) \right]$$

$$g(x,y) = \sqrt{sx(x,y)^2 + sy(x,y)^2}$$

```

  end

```

```

end

```

Figure 9.3 Sobel operator algorithm defined for a subimage.

The next conceptual step of the EGT algorithm is to determine the best threshold values for each subimage. The figure of merit, $M(T)$, is a measure of how well the edges generated by a given threshold, T , match the detected edges. It is computed by each PE for its subimage as follows.

1. For each input-image pixel the local maximum and minimum pixel values in a 3×3 window centered on the pixel are determined.
2. For each possible threshold value (i.e., all gray levels) each input-image pixel is tested to see if it is an *edge point*. It is an *edge point* for those threshold levels greater than or equal to the local minimum and less than the local maximum.
3. The *figure of merit* for a threshold is the mean of the edge-image pixels corresponding to the input-image pixels found to be edge points for that threshold.

The greater the mean of the edge points in Step 3, the better the match between threshold-generated contours and the edges detected by the Sobel operator. To avoid assigning a high figure of merit to a small number of noise pixels, a small constant c can be added to the denominator when calculating the mean, i.e.,

$$M(T) = \frac{\sum_{\epsilon(T)} g(x,y)}{c + \sum_{\epsilon(T)} 1}$$

where $\epsilon(T)$ is the set of edge-image pixels corresponding to edge points for threshold T . This has the effect of decreasing $M(T)$ if only a small number of pixels are above the threshold. Threshold levels are selected corresponding to local maxima in the figure of merit. The number of threshold levels selected

depends on the image; three to six thresholds are not uncommon.

The preceding conceptual steps are embodied in the EGT algorithm shown in Figure 9.4. Let the subimage SI be $r \times r$ and $SI(i,x,y)$ be a subimage pixel, where $0 \leq x,y < r$, $0 \leq i < N$. The algorithm is executed on all subimages (all i) simultaneously and combines the calculations involved in the three conceptual steps where possible to reduce total computation.

Referring to Figure 9.4, the first **for** statement clears the *sumedge* and *nedge* counters. The next pair of nested **for** statements calculates quantities associated with each pixel in the subimage. The first three statements compute the Sobel operator, $g(i,x,y)$. Next, the local maximum and minimum pixel values over a 3×3 window are determined for each pixel. Note that the same pixels necessary for the calculation of the gradient can be re-used in the local maximum and minimum computation. Running sums of the edge-image pixels (gradient values) corresponding to edge points at each threshold (*sumedge*) and a count of the number of edge pixels for each threshold (*nedge*) are updated in the innermost **for** loop. The mean for each threshold (*sumedge* divided by *nedge*) is the figure of merit (*merit*) and is calculated in the final **for** statement using the accumulated sums.

For any threshold-based segmentation scheme, regions may intersect if more than one threshold is chosen per subimage. Note that for any two such regions, the pixels of one must be a subset of the other. For example, an object may be segmented from the background by several threshold levels but with somewhat different perimeters in each case. The action to take when regions intersect can be left to the discretion of the end user of the processed images: it is highly application dependent. Knowledge that regions intersect can be valuable to subsequent object classification algorithms: a set of nested

```

for thresh = 0 to 255 do
  sumedge(i,thresh) = nedge(i,thresh) = 0
  for x = 0 to r - 1 do begin
    for y = 0 to r - 1 do begin
      
$$sx(i,x,y) = \frac{1}{4} \left[ (SI(i,x-1,y-1) + 2*SI(i,x-1,y) + SI(i,x-1,y+1)) \right. \\ \left. - (SI(i,x+1,y-1) + 2*SI(i,x+1,y) + SI(i,x+1,y+1)) \right]$$

      
$$sy(i,x,y) = \frac{1}{4} \left[ (SI(i,x-1,y-1) + 2*SI(i,x,y-1) + SI(i,x+1,y-1)) \right. \\ \left. - (SI(i,x-1,y+1) + 2*SI(i,x,y+1) + SI(i,x+1,y+1)) \right]$$

      
$$g(i,x,y) = \sqrt{sx(i,x,y)^2 + sy(i,x,y)^2}$$

      localmax(i) = max { SI(i,x-1,y-1), SI(i,x,y-1), SI(i,x+1,y-1),
        SI(i,x-1,y), SI(i,x,y), SI(i,x+1,y),
        SI(i,x-1,y+1), SI(i,x,y+1), SI(i,x+1,y+1) }
      localmin(i) = min { SI(i,x-1,y-1), SI(i,x,y-1), SI(i,x+1,y-1),
        SI(i,x-1,y), SI(i,x,y), SI(i,x+1,y),
        SI(i,x-1,y+1), SI(i,x,y+1), SI(i,x+1,y+1) }

      for thresh = localmin(i) to localmax(i) - 1 do begin
        sumedge(i,thresh) = sumedge(i,thresh) + g(i,x,y)
        nedge(i,thresh) = nedge(i,thresh) + 1
      end
    end
  end
for thresh = 0 to 255 do
  merit(i,thresh) = sumedge(i,thresh)/(c + nedge(i,thresh))

```

Figure 9.4 Parallel algorithm for EGT at PE_i

regions might be considered a single object, and a clear classification of any one deemed the classification of all.

9.4.2 Contour Tracing Algorithm

Contour tracing with non-overlapped subimages is more difficult than for the overlapped subimage scenario for two reasons. One arises out of the fact that there can be no assurance that a given object will reside entirely within some subimage. Coping with this fact does, however, allow objects of arbitrary size to be accommodated with no additional effort. The second difficulty arises out of the fact that contour tracing is to be done by multiple processors that must coordinate their activity. Once a one pixel wide border of image data around a given subimage is in place the EGT algorithm requires no further inter-PE communication. Contour tracing can require access to the entire contour by any PE; if that data is not to be duplicated on a truly massive scale, PEs must communicate.

The algorithm is described in a top-down manner. Initial discussion is in general terms and presents the algorithm strategy. Further discussion evolves to levels of greater detail.

The non-overlapped algorithm for contour tracing has two phases. In Phase I, the subimage is segmented within each PE and all local contours (both closed and partial) are traced and recorded. A *partial contour* is a section of a contour with ends at the border of the subimage in which it is located. The partial contours traced during Phase I are connected across subimages to form closed contours in Phase II. Initially, each PE contains the list of threshold values, T_1, T_2, \dots, T_t , selected for its subimage, where t is the number of thresholds. Also, each PE is assumed to have retained those pixels

bordering its subimage that were required by the EGT algorithm. This will allow all contours, closed and partial, within a subimage to be detected, even those that run along the edge of a subimage. A subimage and its border pixels together form an *augmented subimage*.

During Phase I each PE constructs a *contour table* containing an entry for every contour located in its subimage, whether partial or closed. Each entry has the following fields:

- (a) contour identification number,
- (b) threshold value that generated the contour,
- (c) flag indicating whether the contour is closed or partial,
- (d) pointer to the pixel i-x-y coordinate sequence of the contour,
- (e) number of pixels in the contour,
- (f) flag indicating whether the partial contour has been linked,
- (g) address of the PE that linked the contour,
- (h) PE address and contour identification number denoting any partial contour blocking extension of the partial contour described by this table entry, and
- (i) partial contour locked/unlocked semaphore.

Each PE also builds a *partial contour list*. This list has an entry for each partial contour containing the information necessary for connecting partial contours in Phase II recorded under the corresponding contour identification number as used in the contour table (i.e., a pointer to its contour table entry).

Phase I. In Phase I each PE processes its augmented subimage for each of its threshold levels T_i , $1 \leq i \leq t$. Each threshold is processed

independently. With a given threshold selected, the augmented subimage is segmented. All contours, both closed and partial, are traced and added to the partial contour list and/or contour table, as appropriate. Phase I continues until all thresholds have been used in turn.

Contour tracing for a particular threshold begins by searching for untraced contours. This search is carried out by scanning rows of the segmented augmented subimage in a pattern from left to right, beginning with the second row (first row with subimage pixels) and continuing through the next to last row (last row with subimage pixels). When the rows have been scanned, a final scan is made vertically on the second column of the augmented subimage (leftmost column with subimage pixels). Scanning is suspended whenever an untraced contour is encountered, the new contour is traced, its pixels are marked as traced, and then scanning is resumed in order to detect additional contours.

The sequence of background pixels and unmarked object pixels encountered by the scan indicates when an untraced contour has been found. Because any contour pixel must have a neighboring background pixel (by definition of a contour), and background pixels are 4-adjacent, every pixel on a contour has a 4-adjacent background pixel. Thus, whenever the scan encounters an unmarked object pixel within the subimage that has either a preceding or succeeding background pixel, a new contour has been found. The contour pixel at which scanning is suspended is the *start point* of that contour. A contour is assigned an identification number (contour table field (a)) and the threshold in use is recorded (field (b)) when a start point is found. The row scans will detect all contours having at least one pixel with a horizontal 4-neighbor background pixel, even if that background pixel is part of the

augmented subimage but not the subimage. The row scans will miss those contours having only vertical 4-neighbor background pixels, i.e., contours that are horizontal lines extending entirely across the augmented subimage. The column scan will detect these remaining contours, if there are any. In this way, the augmented subimage scanning procedure will detect all contours, closed and partial, within a subimage.

Once the start point for a new contour has been found, that contour is traced. The direction in which a contour is traced has either a clockwise or counterclockwise sense with respect to the object. If a contour is closed within a subimage it can be completely traced by finding successive contour pixels, proceeding in only one of the two directions, and stopping when the trace returns to the start point. For a contour contained only partially within the subimage, tracing in one direction (e.g., counterclockwise) will proceed until the subimage boundary is reached. An *end point* is a location on a contour at which the next object pixel along the contour in the direction of tracing is outside the subimage (i.e., within the one pixel border). In order to complete the trace of the contour within the subimage, it is necessary to return to the start point and trace in the opposite direction. The order in which the directions are taken is immaterial: all contour pixels can be found either way. For this algorithm an arbitrary choice is made to trace counterclockwise first.

In order to define the conventions for tracing, consider the start point to be the center of the 3×3 window shown schematically in Figure 9.5. During a row scan, a preceding background pixel would be at position 4 (see Figure 9.5), and a succeeding background pixel would be at position 0. For the column scan the positions are 2 and 6, respectively. View the contour pixel and its 4-adjacent background pixel as defining a hand of a clock. If this hand is rotated

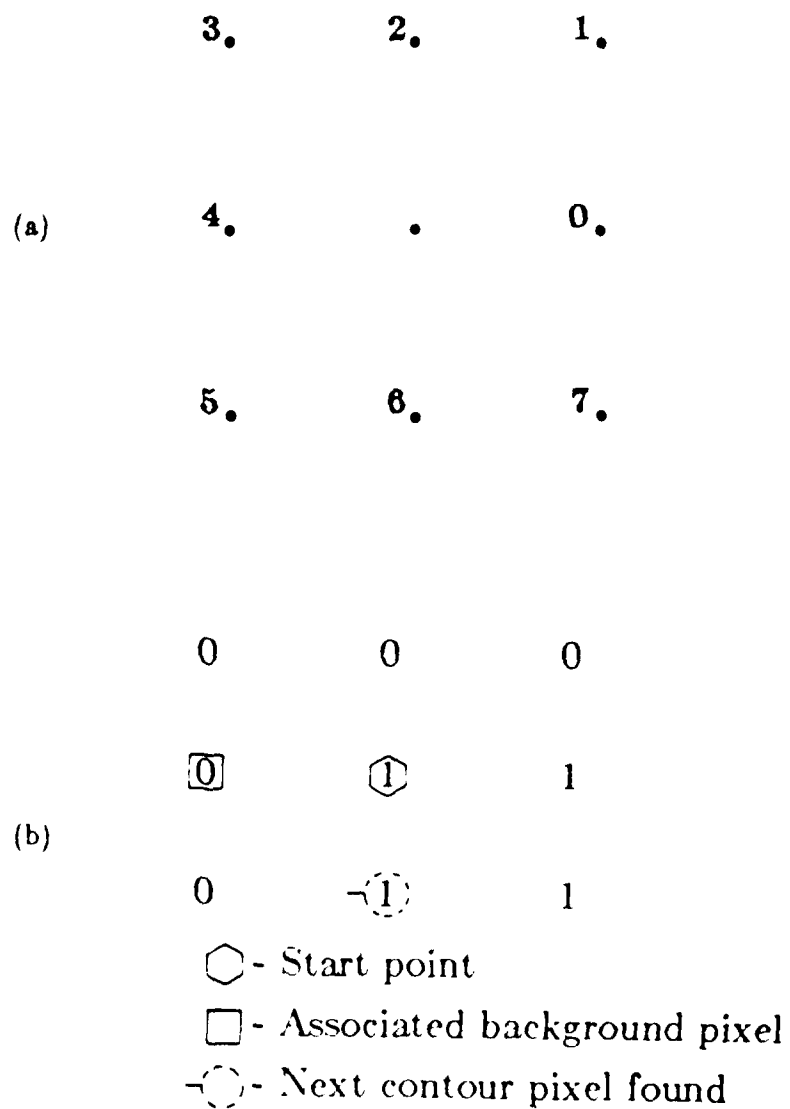


Figure 9.5 (a) Naming convention for the neighbors of the center pixel in a 3×3 window. (b) Example showing start of tracing.

in the augmented subimage then the first object pixel it encounters is guaranteed to be on the exterior of the object (i.e., 4-adjacent to the background), and, hence, a contour pixel. If the hand is rotated counterclockwise, the first object pixel is also the next pixel in a counterclockwise trace of the contour.

The counterclockwise tracing portion of Phase I of the tracing algorithm is as follows. Let p be a position marker, and assign it the value of the position number of the background pixel that is adjacent to the start point. Using modulo 8 arithmetic, increment p by 1 (accomplishing a counterclockwise inspection of the 8-adjacent pixels) and check the corresponding pixel until one of the following conditions is met: the start point is reached, an end point is detected, an unmarked object pixel is found, a marked object pixel is found, or the initial value of p is reached. Conditions are checked in the order listed. Figure 9.5(b) shows an example in which the background pixel was preceding and in position 4. The value of p is thus 4, and incrementing it using modulo 8 arithmetic to the value 5 sweeps an imaginary hand counterclockwise from the 9 o'clock position. As shown in the example, an object pixel will be encountered after incrementing p once more.

If the start point is reached the contour is closed and has been completely traced; the necessary information is stored in contour table fields (c), (d), and (e) and scanning is resumed. Detecting an end point implies the contour is partial. Note that an end point is a just previously traced object contour pixel. The contour identification number and location of the end point are recorded in the partial contour table for use in Phase II, contour table field (c) is set, counterclockwise tracing is terminated, and clockwise tracing begun from the start point. If an unmarked object pixel is found its x - y coordinate is

appended to the contour sequence, it is marked as traced using a bit map, a new value of p is computed, and counterclockwise tracing continues from this new location. The new value of p is $p + 5$ modulo 8 and is the location of the background pixel adjacent the previous contour pixel. Such contours will indicate some, perhaps many, intersecting regions. Finding a marked object pixel implies there is no interior to the object bounded by the contour, at least in the immediate area. Desirable action in this case may range from ignoring such contours to eliminating one pixel wide regions to treating such contours as any other, the end user will choose. If the initial value of p is reached, then all eight neighbor pixels have been examined without finding one in any of the above categories, so the object point is a single isolated point and will be ignored. No entry is made in the contour table, the pixel is marked, and scanning is resumed.

There are some special cases that can arise during the course of tracing. If intersecting regions are of interest to the end application, the bit map can be generalized to one bit map per threshold level, which will make it easy to detect intersecting contours. If isolated points should not be ignored for a particular application, they can be recorded as contours before scanning is resumed.

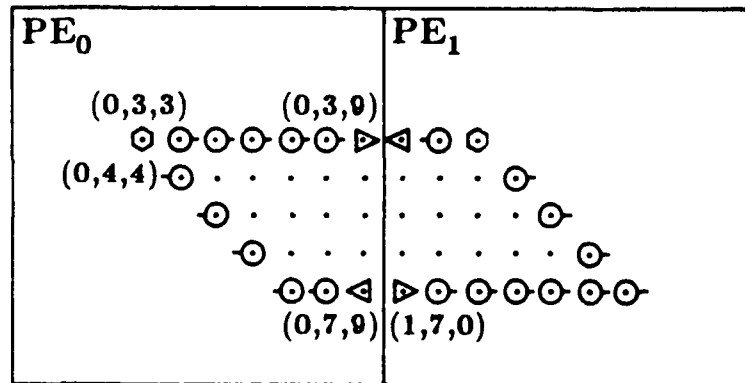
The procedure for clockwise tracing is analogous to that for counterclockwise tracing. The clockwise trace is begun by returning to the start point and restoring p to its value when counterclockwise tracing began. Decrement p by 1 using module 8 arithmetic and check the corresponding pixel until an end point is detected or an object pixel is found. If the second end point is uncovered, the location of that end point is recorded in the partial contour table, clockwise tracing is terminated, and scanning is resumed. If an

object pixel is found its i - x - y coordinate is inserted at the front of the contour pixel list, it is marked as traced, a new value of p is computed as $p-5$ modulo 8, and clockwise tracing continues from this new location. The position $p-5$ modulo 8 is the location of the background pixel adjacent the previous contour pixel for a clockwise trace. There are instances where all pixels of a partial contour will be found during the counterclockwise trace; the clockwise tracing algorithm will correctly confirm this fact if true.

Consider the following contour tracing example based on Figure 9.6. A 10×20 pixel image is divided into two 10×10 subimages; the subimages are loaded into adjacent PEs; and the local threshold value T_1 is applied by each PE. Dots in the figure indicate pixels above the threshold. The one-pixel-wide border around each subimage is not shown.

Each PE_i , $0 \leq i \leq 1$, begins scanning its subimage at pixel $(i,0,-1)$, the leftmost pixel in row 0, the first row of the augmented subimage with subimage pixels. PE_0 locates a start point at pixel $(0,3,3)$ and traces the contour counterclockwise to an end point at pixel $(0,7,9)$. Next, PE_0 traces the contour clockwise from pixel $(0,3,3)$, reaching the other end point at pixel $(0,3,9)$. After the clockwise trace, the first pixel in the i - x - y coordinate sequence describing the contour is $(0,3,9)$. PE_0 resumes its scan pattern at pixel $(0,3,4)$ and finds no other contours in its subimage. Note that, for example, pixel $(0,4,4)$ is not a start point for a new contour at this threshold because it was marked during the trace of the first contour. Meanwhile, PE_1 locates a partial contour with $(1,7,0)$ as the first pixel in its i - x - y sequence.

As a further example, a 30×20 image is divided into six 10×10 subimages; each subimage is loaded into one of six PEs. In Figures 9.7 and 9.8 the results of row scanning and the column scan are shown, respectively.



- ⊙ Start point
- ⊖ Counterclockwise trace mark
- ⊙ Clockwise trace mark
- ◁ End point (counterclockwise)
- ▷ End point (clockwise)

Figure 9.6 Example of Phase I contour tracing for a 10×20 image. The triple (i,x,y) represents the i - x - y coordinates of a pixel.

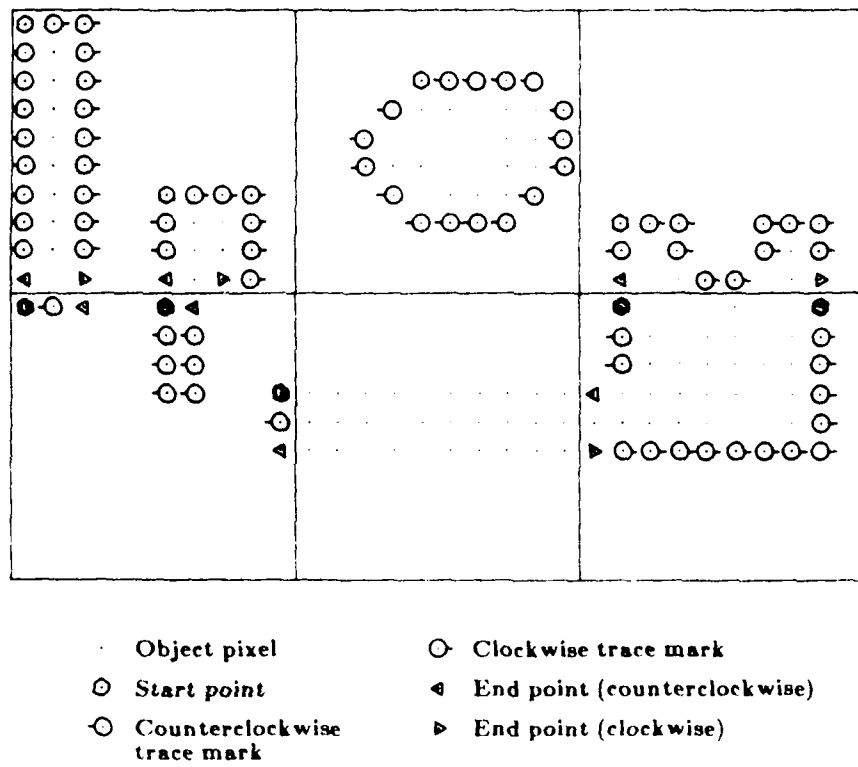
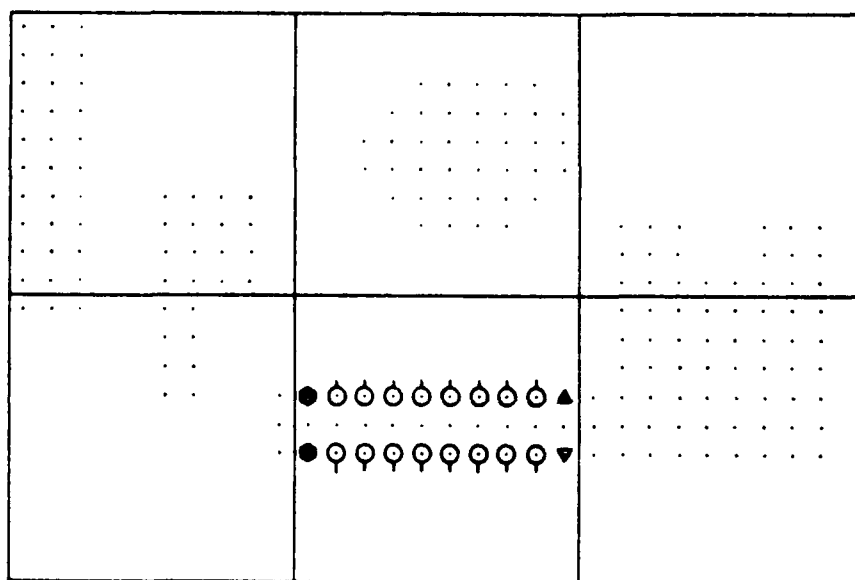


Figure 9.7 Contours found by Phase I row scans for a 30×20 sample image.



- | | | | |
|---|-----------------------------|---|------------------------------|
| · | Object pixel | ⊙ | Clockwise trace mark |
| ⊙ | Start point | ▼ | End point (counterclockwise) |
| ⊙ | Counterclockwise trace mark | ▲ | End point (clockwise) |

Figure 9.8 Contours found by the Phase I column scan for the image of Figure 9.7.

Phase II. In Phase II, partial contours are connected to form closed contours. Two possible conditions to determine when a PE may enter Phase II are: (1) no PE starts Phase II processing until all have completed Phase I, and (2) each PE enters Phase II immediately after completing Phase I. For the latter condition a PE must be restricted to attempt to extend partial contours only into subimages of PEs which are also in Phase II. If all neighboring PEs are still in Phase I, the PE must wait. The first alternative requires time equal to the sum of the times for each phase when phases are carried out sequentially. The second approach may be expected to reduce execution time, because when phases are executed sequentially, the PE with the longest Phase I time need not be the one with the longest Phase II time. However, because the work a PE does is data dependent and can also depend on whether all PEs start Phase II together or not, the second approach may increase execution time in some cases. It is problematic as to which approach is superior.

In Phase II each PE attempts to connect its partial contours to partial contours located in neighboring subimages, extending each partial contour so as to eventually form a closed contour. The goal is for all closed contours to be formed, but only once. Exactly one PE should hold the i - x - y sequence for each contour at the end of Phase II. The question of where to look for extending partial contours must be addressed.

If the end point of a given partial contour is not at a corner of its subimage, then there are three pixels that possibly can extend it, located in the adjacent subimage. That is, there are three locations in the adjacent subimage that are 8-neighbors of the given end point. The PE accesses the partial contour list for the adjacent subimage and considers the possible extending

pixels one at a time in counterclockwise order to determine if any partial contours in the adjacent subimage have the possible extending pixel as an end point. Counterclockwise order is simply the convention chosen. If there is more than one partial contour with the same end point that can extend the given contour, the partial contour that was generated by a threshold value closest to that for the given contour is selected.

To correctly link contours and prevent duplication of linking effort, partial contours can be locked to provide for exclusive access. If an extending partial contour exists, the PE checks the contour table entry pointed to by the partial contour list. If the entry is unlocked and unlinked, the i-x-y sequence for the partial contour is transferred and concatenated to the given partial contour, forming the i-x-y sequence of the extended partial contour. The PE also sets the flag indicating that the adjacent partial contour was linked and leaves its address (contour table fields (f) and (g) of the remote PE). Contour table field (f) notifies the PE containing that partial contour not to expend effort extending that particular partial contour.

If the end point of a given partial contour is a corner point of its subimage there are five pixels located in adjacent subimages that can possibly extend the contour. Since these five pixels are located in three different subimages, the PE attempting to extend the given partial contour must check for continuation in each of the adjacent subimages in a counterclockwise order; pixels within a subimage are checked in counterclockwise order.

Note that regardless of where end points lie, the search for pixels to extend the partial contour can be widened to allow for segmentation-induced contour discontinuities at subimage boundaries. Thresholds could be interpolated across subimage boundaries to allow partial contours with non-

adjacent end points to be joined. Assume that PE_i has a partial contour to extend. If no suitable partial contour is found in the partial contour list of the adjacent subimage, PE_i probes into the adjacent subimage to determine if an extension of the partial contour can be generated by the threshold, T , used by PE_i to trace its partial contour. If so, PE_i extends its partial contour by accessing the data from the adjacent PE. Instead of creating an entire segmented subimage for the threshold T , PE_i dynamically thresholds pixels as needed. This contour generation using T is done since it is possible that the partial contour in the adjacent PE was not located in Phase I because different threshold values were used.

A lone object pixel located adjacent to the border of a subimage will be listed as a partial contour during Phase I. During Phase II such a pixel will either be linked to a partial contour from an adjacent PE or identified as an isolated point. Isolated points can be detected by their length as partial contours and the lack of extending contours. Once detected they can be treated as other isolated points.

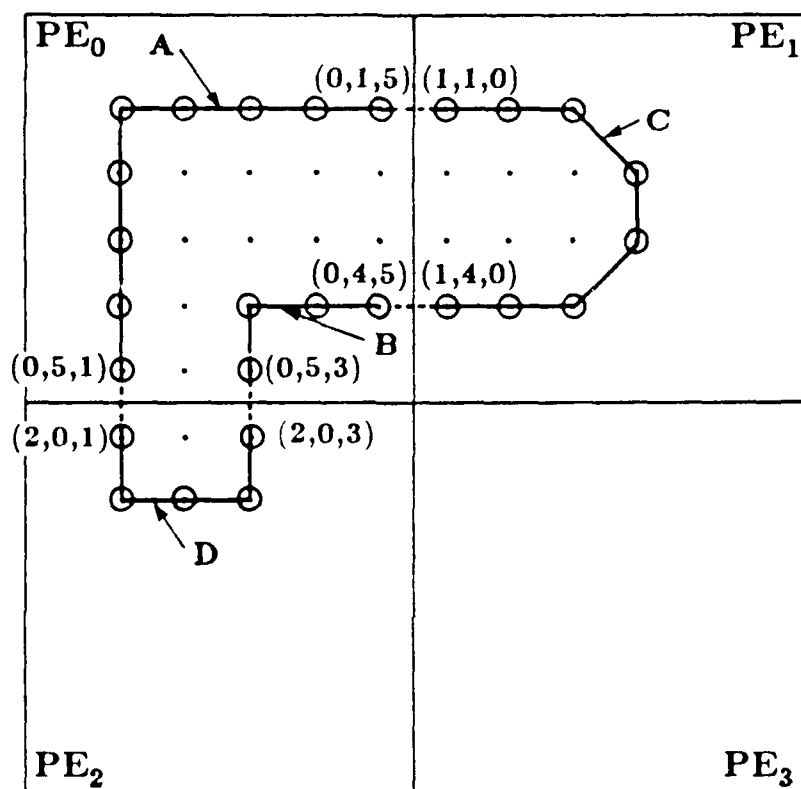
Once PE_i locates a partial contour in an adjacent subimage that continues the given partial contour and has stored the concatenated contour, it repeats the process, as necessary, by following the contour to the next PE until the contour is closed or cannot be extended. A limit is placed on contour length to guarantee algorithm termination in the event of a pathological image; contour length can be determined from contour table field (c).

Because many processors are each working independently, yet their actions must be coordinated to reach the goal of extracting contours, knowledge of where partial contours can be extended is insufficient. There must be a protocol to guide the activity and necessary support mechanisms must be in

place. By way of illustration consider the example in Figure 9.9 where a 12×12 pixel image is divided between four PEs. After Phase I, PE_0 has found partial contours A with end points (0,1,5) and (0,5,1) and B with end points (0,4,5) and (0,5,3); PE_1 has found partial contour C with end points (1,4,0) and (1,1,0); and PE_2 has found partial contour D with end points (2,0,1) and (2,0,3).

Assume PE_0 attempts to extend A (A must not be marked as linked in field (f)) from pixel (0,1,5). It may well be that A is the first entry in the partial contour table of PE_0 , and pixel (0,1,5) is the first location in its i-x-y sequence; partial contours may be considered for extension in any order and extended from either end, however. If PE_1 attempts to extend C at this time, the danger that a given contour (ACBD in this instance) can be formed more than once by multiple PEs is clear. Thus, a mechanism to prevent one PE from using a contour table entry while another PE is in the process of using that entry must be available.

A *semaphore* is a variable, the value of which indicates whether or not a critical section can be entered [Dij68] and can be used to enforce exclusive access to data. There is a semaphore for each contour table entry, which can take on a value of zero or one (field (i) in the contour table). Before a PE enters a critical section for a given contour, it performs a *P-operation* [Dij68] to determine if the semaphore for that contour is one, indicating it is unlocked. If so, the processor sets the semaphore to zero, locking the contour table entry so that no other processor can access it, and enters the critical section to link the new partial contour. When the PE reaches the end of the critical section, it performs a *V-operation* [Dij68], resetting the semaphore to one and unlocking the contour table entry. If the *P-operation* had determined that the



⊙ Pixels traced in Phase I

Figure 9.9 Example to illustrate Phase II activity, protocol, and mechanisms. End point coordinates are given where (i,x,y) represents the i - x - y coordinates of the pixel.

semaphore was zero, the PE receives a message indicating that the partial contour is locked.

Returning to the example of Figure 9.9, PEs now first lock a partial contour they will try to extend. Thus, when PE_0 attempts to extend A it will receive a message that C is locked. If PE_0 waits for C to become available, then eventually PE_0 , PE_1 , and perhaps PE_2 , will all be waiting for access to partial contours that another member of the group has locked.

The situation when each of two or more processes are halted while waiting for the other(s) to continue is known as *deadlock* [Sto80]. In MIMD mode each PE executes a distinct process, so if PEs wait on each other it is deadlock. If a PE is blocked due to a locked partial contour then deadlock is prevented if (1) no PE may wait for access to a locked contour table entry of another PE, and (2) all PEs must unlock blocked partial contours. This deadlock avoidance scheme is one aspect of the contour tracing protocol needed for Phase II.

When a PE successfully extends a partial contour it performs the appropriate modification on the (f) and (g) fields of the contour table containing the extending partial contour. Once again considering Figure 9.9, assume PE_1 has formed CB and PE_2 has formed DA, both have modified contour table entries as needed, and are ready to continue to extend their partial contours C and D, respectively. If both try to extend simultaneously, then both will abandon their partial contours, and the closed contour will not be found. The remaining aspect of the contour tracing protocol for Phase II addresses this problem.

To insure that the linking of a contour will not be abandoned by all PEs, the following protocol is used. A *total ordering* is established on the PEs.

Given a binary relation R on every pair of elements a and b in set S , if either aRb or bRa then R totally orders the set S . For example, \leq totally orders the set of all integers. One convenient total ordering is the \leq relation on PE addresses; clearly, others are possible. Assume PE_i is blocked from extending a partial contour X because PE_j , which has higher priority (i.e., $i \leq j$), has locked Y , the partial contour PE_i needs. In that case, PE_i unlocks X and sends a message informing PE_j that PE_i has abandoned its attempt to extend X . The message sent from PE_i to PE_j contains the identification numbers of X and Y and the value i . After receiving the message, PE_j searches its contour table to determine if it abandoned Y . To do this it uses field (h) of the contour table. If it had, it now begins extending Y by adding X from PE_i (unless X has been locked in the interim by yet another PE tracing a portion of the contour comprised in part by X and Y) and continues until it has closed the contour or is blocked.

The deadlock avoidance and PE priority protocols allow PEs to attempt to extend their partial contours in any order and from either end point with correct results assured. This is advantageous if a long contour extending across many subimages is to be formed during Phase II from its constituent partial contours. Many PEs can all work simultaneously to extend partial contours along this contour, each dropping out to work on any other partial contours that might be in its subimage as it is blocked by a higher priority PE. In effect, contour tracing is performed using the divide-and-conquer model at both the image level (subimage per PE) and the contour level (possible simultaneous, multiple-PE, partial contour linking for a single contour).

When Phase II of the algorithm is complete, the i - x - y sequence for each contour in the image will be stored in the memory for exactly one of the PEs

that contained part of the contour originally. For the example of Figure 9.9, PE_2 will contain the i-x-y sequence for contour DABC in the event it and PE_1 tried to simultaneously extend CB and DA, respectively, since it has higher precedence under physical address ordering. Because each PE tries to connect its contours independently, there is no simple rule to determine which PE will finally close a given contour. Although this may be undesirable in some instances, a side effect of the procedure is to tend to equalize the processing load of the PEs and the number of closed contours found by each PE.

9.5 Example of Non-Overlapped Method Performance

The parallel algorithms described in the previous two sections have many potential applications, one of which is the visual inspection of thick film products within an industrial environment. In such an environment scene illumination can be closely controlled, so subimage-based EGT may not be needed for its adaptive capability, but its accuracy may be desirable. However, using subimage-based EGT incurs no penalty. A functional simulation of the algorithms of the previous sections has been coded and executed; the results are presented in this section [MiD83].

The section of a printed circuit board in Figure 9.10 is shown with a 16×16 pixel subimage grid, the subimage size used in the simulation. The board was purposefully illuminated unevenly to illustrate the capability of subimage-based EGT. When a single best global threshold is used for the image, the result is the binary image shown in Figure 9.11. Clearly, a single threshold level will not yield satisfactory segmentation of the circuit wires and substrate. The EGT algorithm was used to determine a suitable threshold for each subimage, since ideally this image would have pixels of only two values, a

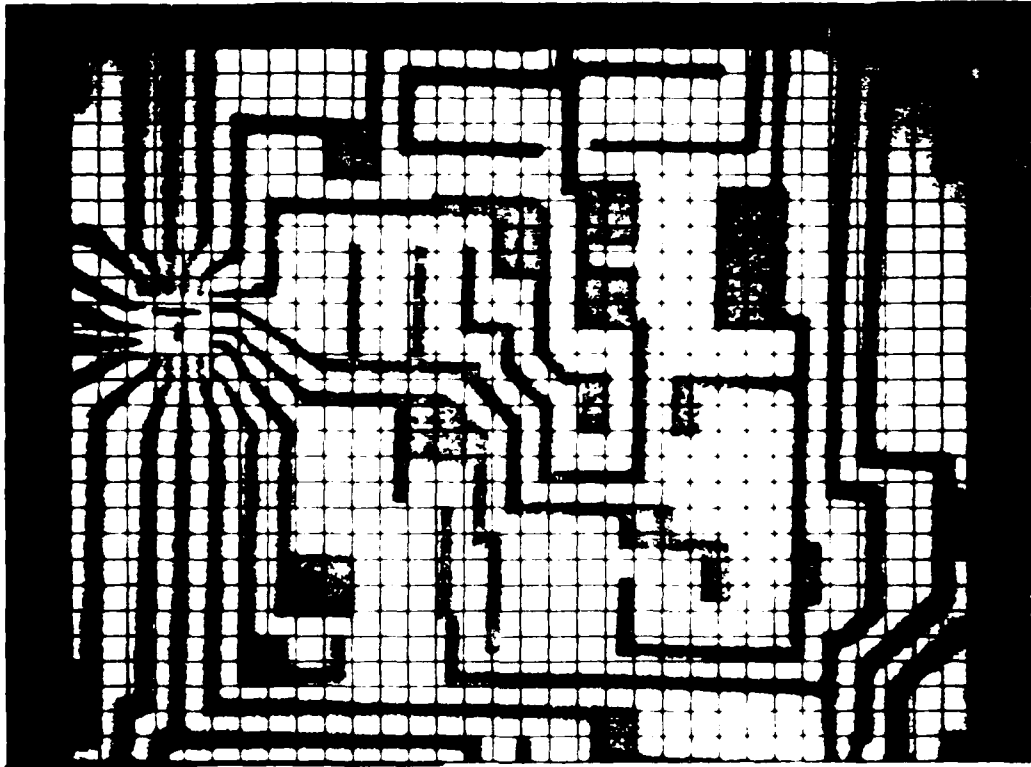


Figure 9.10 Section of a poorly illuminated circuit board overlaid with a 16 x 16 pixel grid

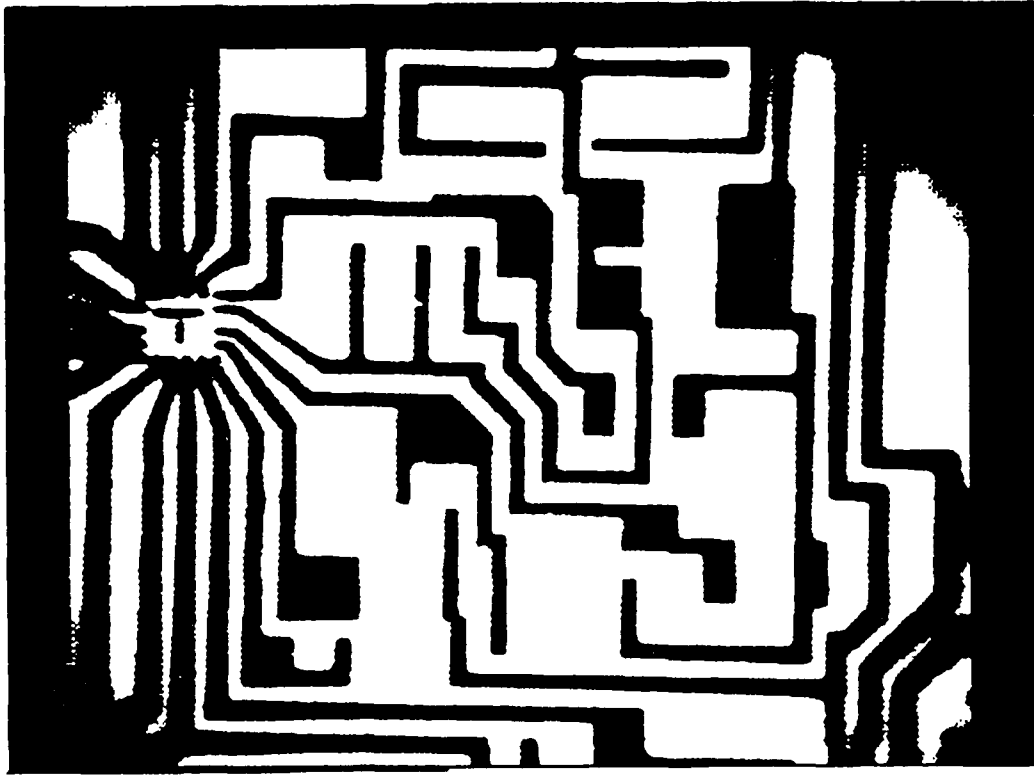


Figure 9.11 A binary image obtained by thresholding Figure 9.10 with a single threshold of 153.

single threshold in each subimage suffices. In addition to using an additive constant to suppress noise when computing threshold figure of merit values, the resulting merit value functions $M(T)$ were smoothed over five points to yield the desired single maximum. The resulting merit value graphs for a representative group of the subimages are shown in Figure 9.12.

The EGT algorithm becomes more adaptive as the subimage size decreases, but choosing too small a subimage can create problems. For instance, there may no longer be an edge within a subimage, causing the algorithm to select a threshold related to the noise or DC change within the subimage. There are some subimages in the example image that do not contain an edge. In these subimages small gray level variations cause the EGT algorithm to detect false merit value peaks. It was necessary to apply a threshold to the merit values themselves before an edge was considered detected for a subimage.

The threshold used to insure that a chosen merit value peak actually corresponds to a significant edge should be related to the distribution of the Sobel values for the entire picture. In the case of the thick film board, both the Sobel-generated edge image and input image histograms should be bimodal, but due to the poor lighting, the modes are smaller in magnitude and wider, and may even be merged. A percentile of the Sobel values over the entire image is a very robust merit value threshold. Empirically, the 50-th percentile of the Sobel value histogram is an appropriate merit value threshold for this example. Any value between the 30-th and 70-th percentile gives equivalent results (see Figure 9.12).

Consider the problem of assigning gray levels to the degenerate subimages that do not have merit values above the merit threshold. Neighboring

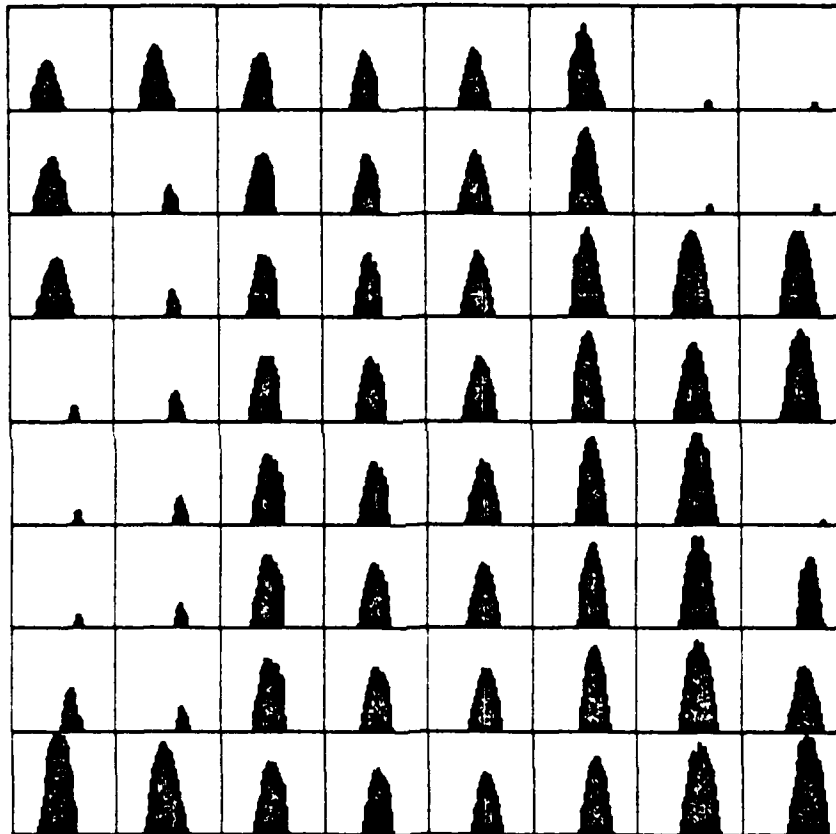


Figure 9.12 The EGT merit value graphs for the 64 subimages in the upper left corner of Figure 9.10. The horizontal axis for each graph is gray value (0 to 255) and the vertical axis is the threshold merit value (0 to 32).

subimages at increasing distance are searched for the nearest with a valid threshold. Once found, this valid threshold is used to threshold the degenerate subimage. Table 9.1 shows the figures of merit, corresponding thresholds, and the thresholding decision made for subimages in the upper left corner of the image. The first number in each box is the maximum figure of merit value, the second is the corresponding gray level threshold value, and the third is an asterisk if the corresponding gray value is a valid threshold, i.e., figure of merit value above Sobel 50-th percentile. A "W" or "B" as the third entry indicates that a valid threshold was not found for that subimage, and the default value was chosen as white or black, respectively. No completely black subimages were found in this portion of the image, thus there are no "B" entries in this table. The segmented image using the results of EGT is shown in Figure 9.13. Compare this with the image segmented using the best single threshold shown in Figure 9.11.

The actual partial contour tracing and linking is straightforward because segmentation is binary in the example. The black wire traces are considered to be 8-neighbor objects, and the white substrate a 4-neighbor background. Contour tracing was as described in the previous section. Every partial contour in the example was found to have an end point that was an 8-neighbor to an end point of a partial contour in the adjoining subimage. The found contours are shown in Figure 9.14. Except for inadequate resolution in the input image near the left edge, all contours were correctly extracted.

Table 9.1 Computed parameters for the figure of merit values shown in Figure 9.12.

11.51 94 *	14.86 102 *	12.93 107 *	13.47 104 *	14.02 117 *	19.23 133 *	2.23 183 W	1.83 187 W
12.91 111 *	6.63 139 W	13.66 112 *	13.53 109 *	14.25 120 *	19.47 140 *	1.86 186 W	1.98 191 W
13.65 119 *	6.39 145 W	14.05 117 *	14.64 116 *	14.81 123 *	20.16 141 *	19.79 140 *	19.43 149 *
3.41 160 W	6.84 154 W	14.78 119 *	14.67 118 *	14.79 126 *	20.40 142 *	18.11 144 *	20.90 146 *
3.13 165 W	6.58 160 W	16.27 114 *	14.44 118 *	14.76 128 *	20.00 147 *	21.01 142 *	1.26 201 W
2.93 167 W	5.82 162 W	16.77 117 *	15.18 120 *	15.04 134 *	19.65 150 *	21.40 158 *	16.48 167 *
9.91 149 *	5.76 165 W	16.94 113 *	14.91 123 *	14.62 149 *	19.86 155 *	21.46 144 *	15.18 156 *
23.70 111 *	21.28 112 *	16.92 120 *	15.57 131 *	14.67 136 *	17.99 153 *	21.34 151 *	23.47 160 *

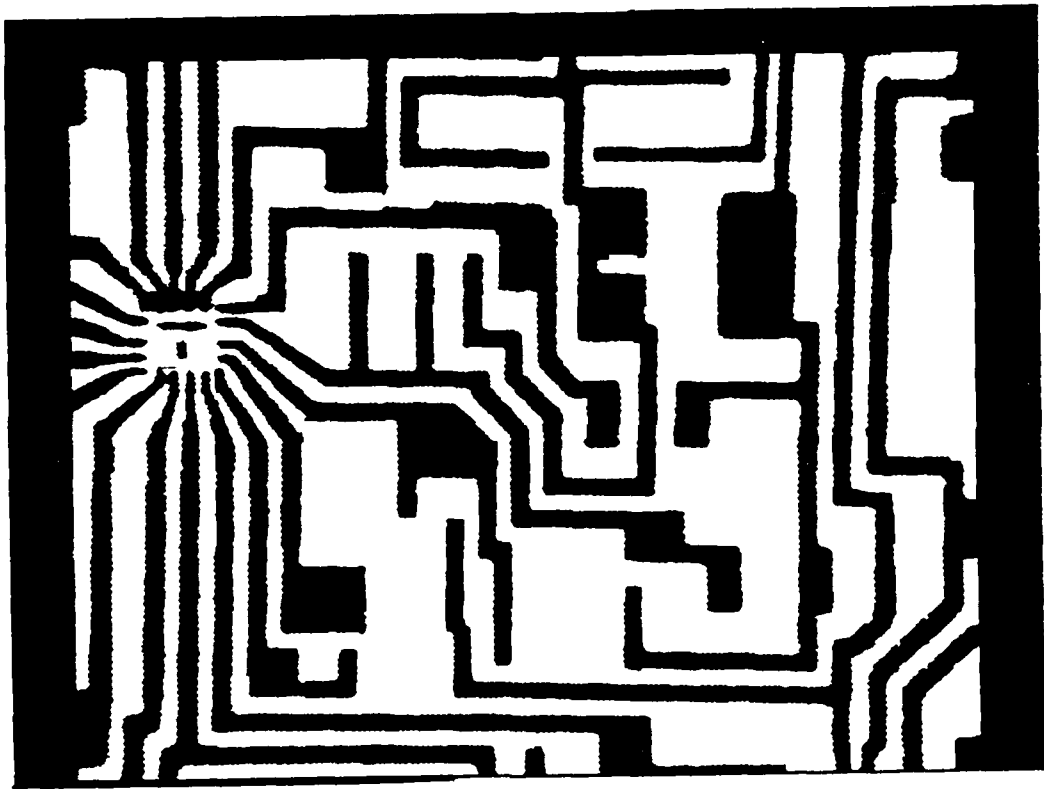


Figure 9.13 Binary image resulting from EGT-based segmentation of Figure 9.10.

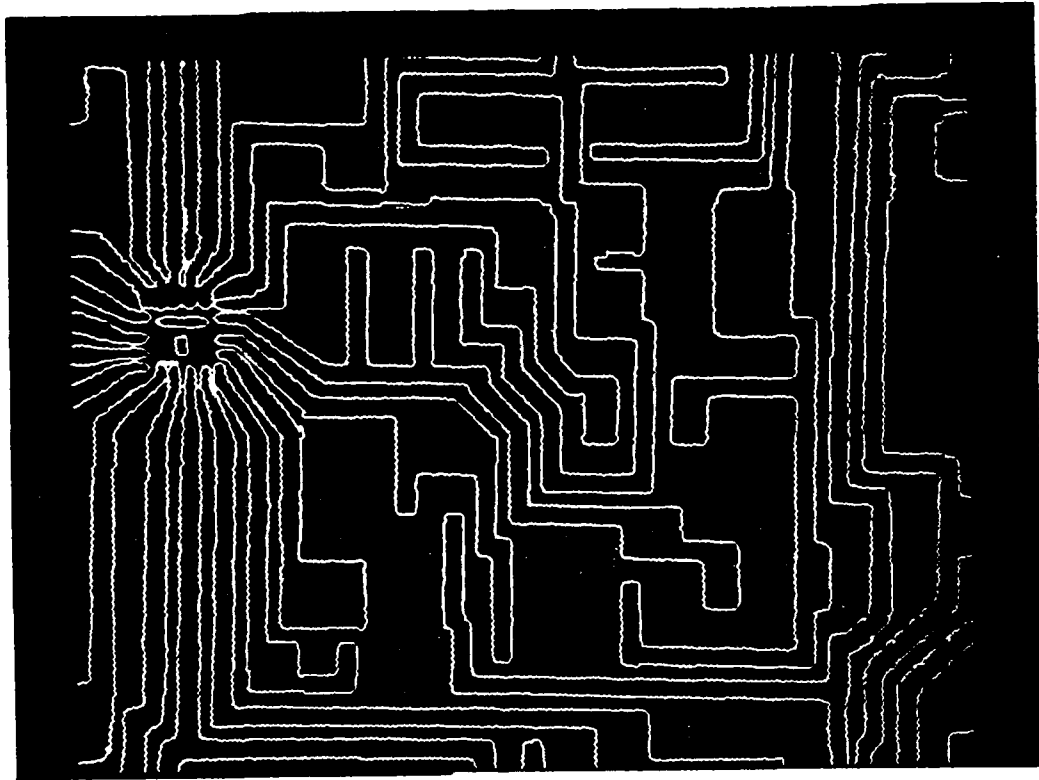


Figure 9.14 Contours extracted from Figure 9.10.

9.6 Analysis of Non-Overlapped Method

Choice of Parallel Architecture. Details of the computation in each PE for the EGT algorithm (Figure 9.4) are quite similar. This is particularly true during the Sobel edge image generation and determining the local minima and maxima. Thus, a single instruction stream will be relatively efficient. SIMD mode operation facilitates the PE-to-PE communication necessary when subimage border pixels within each PE must be processed. Transmission delays incurred due to PE-to-PE data transfers can be overlapped with data processing to reduce total execution time.

PE-to-PE communications in MIMD mode require explicit synchronization between the two processors transferring data. Thus, SIMD mode transfers should be used when possible for large transfers of data to efficiently provide each PE with the one-pixel-wide border points of its augmented subimage. However, once each PE has all of the data it needs to perform the EGT algorithm, the calculations could proceed in MIMD mode.

There are two aspects of the EGT algorithm less well matched to SIMD parallelism. One is the square root calculation in the Sobel operator. Square root is usually performed in software with an iterative, data-dependent execution time algorithm. There will be lost processor time for synchronizing after the operation. The other is the fact that each PE performs the innermost **for** loop in Figure 9.4 using a different *localmin* and *localmax* and, hence, performs the statements in the loop (updates the sums) different numbers of times. PEs are disabled when they finish their looping, since PEs must remain synchronized in SIMD mode. The total time to perform the innermost **for** loop is the maximum time taken by any PE.

Although MIMD mode would make the execution of the innermost **for** loop (Figure 9.4) more efficient (because no PEs would be disabled), this advantage must be weighed against the extra time involved in switching from SIMD to MIMD mode and requiring that each PE perform its own control flow operations for the outer two **for** loops. Control flow operations include initialization and incrementing of loop counters, evaluation of conditional expressions, and branching. These operations are performed by a system control unit in SIMD mode for the outer loops and can be overlapped with the PE operations. Overall, SIMD mode execution seems more advantageous for the EGT algorithm.

The number of PEs to use is an important consideration for the implementation of contour extraction on a parallel computer. Consider first using few PEs. As the number of PEs (N) is decreased, subimage size increases for a fixed-size total image size. For large subimages, the ratio of subimage border pixels to total pixels is low. This increases processing efficiency because inter-PE transfers of border pixel values take up only a small fraction of the total time. The *speedup factor* is the ratio of execution time on a serial computer of power equivalent to one PE to execution time with a parallel computer of N PEs. It approaches N for arithmetic operations for large subimages. A speedup factor of N is considered optimal because it indicates that N PEs are accomplishing N times the work of a single PE in a given time. However, if image average brightness is markedly uneven, subimage size cannot be arbitrarily large without compromising segmentation accuracy. Also, very large subimage size may potentially degrade performance due to the finite size of physical main memory in any computer; for better performance, subimage size should be restricted so as to avoid paging while processing a subimage.

Consider now a large number of PEs applied to the contour extraction problem. As N is increased, the subimage size decreases. In this case, the ratio of border pixels to total subimage pixels is high and inter-PE transfers account for a large percentage of total processing time. Total processing time is minimized as N increases, however, the speedup factor decreases (much time is spent on transfer operations between PEs, which are not needed for the serial machine). Too large a value of N will result in too small a subimage size for good accuracy in the EGT step. Very small subimages may contain no true contours yet report false ones because of image noise. Also, use of a large number of PEs may be inefficient for contour tracing if it reduces the number of complete contours found in Phase I, requiring heavier use of inter-PE communication to close contours in Phase II. Note, however, that if most objects span many subimages few closed contours will be found in Phase I, regardless of subimage size (recall the circuit board example).

This conflicting demand on N clearly shows the value of considering the algorithms comprising contour extraction as a whole, rather than as a sequence of individual, independent algorithms. Performance measures that quantify various demands [SSS82] can aid in determining the value of N to use. The value of N should be chosen given knowledge of the specific performance characteristics of a particular parallel processor implementation.

The activity during contour tracing is strongly dependent on the details of the input image; each PE has a set of contours and partial contours that is in all likelihood unique. Phase I of contour tracing requires only locally available data, and execution time is data dependent. Phase II makes heavy use of non-local data and also has data-dependent execution time. Linking contours is feasible only as an asynchronous task in which each processor proceeds with

regard to its circumstances. Both phases of contour tracing are suited to MIMD mode. The fraction of the total execution time for contour extraction spent on contour tracing is typically small.

Comparison to Overlapped Approach. The advantages of the overlapped approach presented in Section 9.3 versus the non-overlapped approach are twofold. One, partial contour extension is not necessary. This greatly simplifies the contour tracing process. Two, only sufficient primary memory to hold one subimage for processing is needed.

The disadvantages of the overlapped approach are threefold. First, the maximum size of an object of interest must be known *a priori* so that subimage size can be established. Subimage size is constrained by the fact that EGT performance tends to degrade with increasing subimage size relative to image size. Thus, there will be a practical limit on the maximum object size. Also, an object may happen to be completely within more than one subimage. This must be recognized and handled appropriately. Second, thresholding (including EGT) tends to perform less well when objects are small relative to the image (in this case, subimage) size. The overlapped method requires that objects cover no more than one fourth the area of a subimage. Thus, there is a practical limit to the range of object sizes that will be handled well for a given image; objects much smaller than the largest objects are at a disadvantage. Finally, each pixel is processed for contour extraction four times, once for each of the four overlapping subimages to which it belongs.

The non-overlapped scenario could be implemented on a serial computer system with virtual memory [Den70]. The disadvantage of this approach is that when a contour spans more than one subimage, linking partial contours requires that a representation of the subimages with partial contours to be

linked, as well as the associated contour table information, be accessible. This may result in significant delay due to paging subimages into primary memory. Paging overhead does not occur on a parallel system since the entire image is stored in primary memory. Thus, it is the multiplicity of primary memories (the large primary memory space) in a parallel system such as PASM that makes the non-overlapping subimage approach practical.

The disadvantages of the non-overlapped method are that contour tracing is more complex and sufficient memory to process an entire image is needed. The advantages of the non-overlapped algorithms are fourfold. There is no limit to maximum object size, each pixel is processed just once, threshold choices may be made more precisely because subimages that are small relative to the largest objects can be readily used, and they are well-suited for parallel processing.

9.7 Implications for Network Design

When subimage border pixels are needed for the EGT algorithm, all PEs simultaneously request the same border pixel relative to their subimages. For example, when EGT begins (with the upper left corner subimage pixel) all PEs will request (from the PE to their upper left) the pixel immediately above and to the left of their upper left corner pixel. This data transfer can occur for all PEs simultaneously. For contour tracing, the communication needed is between 8-neighbors PEs and arbitrary others. These communication links must be established to support partial contour linking. Hence, the interconnection network should be able to perform permutations involving the 8-neighbors of a PE as well as any one-to-one connection.

Because both SIMD and MIMD modes of processing are involved in the contour extraction task, the interconnection network should be able to function well in both modes. Networks that are controlled by routing tags, or in some other distributed manner, can more easily operate in MIMD mode than centrally controlled networks. This is particularly true when the patterns of interconnection change rapidly, as is the case in Phase II contour tracing.

The PE-to-PE transfer of information must be efficient or the system will be slowed. In its simplest form, this communication would be handled by the PEs, each PE would control the network and preform all the network protocol support. The routing tag control scheme for the ESC, for example, would be easy for the PEs to implement.

Each word transferred and each new network setting can require processor instructions. A more efficient method of PE-to-PE communication is by *direct memory access* (DMA). DMA is a method for storing or retrieving data without processor intervention. In its basic form, a PE would enter a DMA handling routine upon request from another PE. This routine computes the local memory address range of the data satisfying the request (e.g., partial contour *i-x-y* coordinates), sends this information to the special DMA hardware and sets the network as needed for the transfer. The DMA hardware then would autonomously retrieve the information from local memory and perform the necessary network interfacing to send the data to the requesting PE. DMA hardware is often designed to operate on a cycle-stealing basis so that PE access to local memory is not severely affected. The PE is still responsible, however, for checking the incoming data (after transfer is complete) for transmission errors and so forth.

A more advanced implementation of DMA capability is the use of an intelligent *network interface unit* (NIU). Requests for data from remote PEs would be made to the local NIU, which would interpret and satisfy the request by coordination with a remote NIU. The NIU would combine DMA capability with network protocol support. VLSI technology may allow ready fabrication of sophisticated NIUs.

Consider the transfer of subimage boundary points, and for the sake of example, let $R = 4096$, $N = 1024$, and $R/\sqrt{N} = 128$. Rather than involve the source and destination PEs of each transfer in that transaction for each pixel, one of the DMA modes just described would be of great use. If pixels were stored in PEs by row (rows numbered 0 to 127), and transfer from PE_i to $PE_{i+\sqrt{N}}$ was selected, the DMA hardware of PE_i would be instructed to transfer 128 pixels starting at the address of row 127 of the subimage. The DMA hardware associated with $PE_{i+\sqrt{N}}$ would be set to read 128 pixels from the network and store them beginning at an address representing row -1, relative to the subimage of $PE_{i+\sqrt{N}}$. When data is transferred from PE_i to PE_{i+1} , the situation is more complicated in that pixels to be transferred are not stored contiguously in the memory of PE_i . Conventional DMA hardware only supports physical block transfers of data. Here a strong case for an intelligent NIU can be made: an NIU could accept more complicated instructions such as "transfer 128 pixels starting at address X, taking every 128th pixel."

The interconnection network and any DMA or NIU hardware would be heavily used in Phase II processing when PEs extending partial contours probe remote PE memories that may contain the extensions of the partial contours. As in the EGT algorithm, the NIU would be of great use because it could

process queries about possible extensions to partial contour without interrupting the remote PE. There would be a combination of short and long messages between PEs during this phase. A short message would occur when a PE extending a partial contour requests information about possible extending partial contours from a remote PE. If a connecting partial contour is found, a long message consisting of the $i-x-y$ sequence of the partial contour would be sent. Thus, the interconnection network should support a variety of message sizes so that the efficiency of sending either type of message is high.

9.8 Implications for System Design

The EGT and contour tracing algorithms were found to be clearly suited for SIMD and MIMD mode processing, respectively. Probably the most basic requirement for the operating system and the system hardware is to allow a single job to switch dynamically between SIMD and MIMD modes of operation. With only SIMD capability, vast inefficiency would occur in later stages of the scenario. Having only MIMD mode is a less serious handicap, but will lengthen execution time for the Sobel operator and determining local maxima and minima, due to the need for explicit synchronism and data sharing, and the inability to perform loop control in a central control unit while PE computation occurs. Thus, the capability to switch dynamically between SIMD and MIMD modes is important so that each subsequent portion of the scenario can be executed in the most appropriate operational mode.

The PEs of an MIMD parallel computer must be able to operate independently on their own data and instruction streams, and so are commonly thought of as complete computer systems in their own right with input/output circuitry befitting a working environment in which each PE is closely coupled

to others. The appropriate nature of the processor in an SIMD system PE is a more open question. At one extreme, a PE could consist of only an arithmetic/logic unit (ALU), a microcode-level interface to the system control unit, basic input/output circuitry, and an interface for processor status information. At the other, the SIMD PE would have the functionality of the MIMD PE, but would lack instruction fetch capability and would receive machine language commands from its control unit rather than from its local memory. Of course, a PE can combine all the capability of a pure MIMD PE with the ability to disable the program counter and accept external machine language instruction streams. Such a PE design would be suitable for SIMD/MIMD systems, including PASM. It might be thought that frequent operation of such a PE in SIMD mode might underutilize its capabilities. Study of the EGT algorithm shows that this is not necessarily true, i.e., executing EGT in SIMD mode enhances overall performance.

Considering the EGT algorithm, there are two steps with data-dependent times: computing a square root for the Sobel algorithm and the innermost **for** loop (see Figure 9.4). Removing the variability and/or shortening the execution time of these steps would improve the SIMD execution of the algorithm. In the case of the square root operation, a fixed time solution either in the form of a new, data-independent algorithm or special PE hardware support is one answer. The algorithm could be executed by having a system control unit issue the instructions, or the algorithm could be embedded in PE microcode, giving each PE a "square root" instruction. Another answer would be to use a modified Sobel operator formulated without the square root operation: accuracy considerations might rule out this option.

The innermost **for** loop can probably benefit most from reducing, rather than fixing, execution time. If each PE could maintain its own loop counters instead of relying on a control unit, execution time could likely be reduced. Incorporating local index registers in each PE would support this. Because PEs will execute this loop different numbers of times, as well as for different ranges of loop index values, there must be a mechanism to disable PEs that finish the loop before others. One method to do this uses processor address masks [Sie77a].

Two alternatives were presented for determining when a PE can enter Phase II. With the first, PEs are allowed to start Phase II processing after all have completed Phase I. Hence, the operating system and hardware communication paths must allow for synchronization of the PEs within an MIMD job.

Since semaphores play a large part in ensuring correct linking of partial contours in Phase II, processors should be able to perform a "test-and-set" operation to facilitate correct semaphore implementation. Test-and-set is an uninterruptible, or *atomic*, action in which the value of a data item in memory is read, compared, and, upon meeting the test of the comparison, a new value is stored in place of the original data item value. While methods to implement such a facility are known for sequential processors, the parallel processing environment, in which the item to be tested may be remote to the PE, presents a greater challenge. An intelligent NIU at the destination PE could perform an atomic "test-and-set" instruction for a requesting PE.

Once all PEs have completed Phase II of the contour tracing algorithm, the job is done. In order for the PASM System Control Unit to know that the job has been completed, each PE must be able to signal its Micro Controller

when it has finished, and MCs must be able to signal the SCU.

9.9 Conclusions

A number of observations were made and conclusions drawn from the contour extraction scenario. In particular, non-overlapped formulation of the algorithms for parallel processing was shown to lead to several advantages including speedup, elimination of object size constraints, and potential for improved accuracy. Necessary protocol and mechanisms to support their correct execution were detailed. Overall, the parallel algorithms presented are strong contenders to replace previous methods in some applications.

One such application is quality control inspection of printed circuit boards. Large object handling capability is needed for following long circuit traces, and sufficient speedup is necessary for timely response. Other applications include those involving military environments in which real-time processing capability is crucial.

Considering an entire scenario in the light of parallelism is a useful approach for matching image processing tasks and parallel architectures. Operating system features found to be important and/or useful include support for

1. dynamic SIMD/MIMD mode switching,
2. synchronization within MIMD processes,
3. semaphores, and
4. access to the local memories of one PE by another PE.

Necessary and/or desirable hardware features include

1. an interconnection network that supports permutation connections between 8-neighbor PEs and direct connection between non-8-neighbor PEs,
2. an interconnection network equally effective in SIMD and MIMD environments (implies decentralized network control) and with widely varying message lengths,
3. an intelligent network interface unit,
4. a test-and-set instruction,
5. PE designed for constant-time instructions and fast loop index manipulation,
6. provision for PEs to signal job completion to a system controller, and
7. provision for selective PE enabling/disabling.

The stated operating system and hardware features are consistent with the capabilities and design of both the ESC network and PASM.

CHAPTER 10

CONCLUSIONS

This research has addressed three main topics in the area of parallel/distributed processing computer systems. Specifically, a fault-tolerant multistage interconnection network was introduced, the performance of alternative switching element designs suited for VLSI implementation was analyzed, and a parallel scenario for digital image contour extraction was studied. These three topics are unified by their joint contribution to the engineering knowledge necessary for parallel/distributed computer system design.

The demand for very high-speed computing motivates this entire area of research. There are many disciplines with important problems that can neither be feasibly solved given the performance level of available computers, nor be solved without computers.

A sample of parallel processing computer research projects was presented in Chapter 2. The PASM multimicroprocessor system was among those described. Unique features of PASM include being (1) dynamically reconfigurable, (2) able to operate in either SIMD or MIMD mode of parallelism, and (3) able to be partitioned into machines of different sizes, each of which can operate in SIMD or MIMD mode. Each of the systems surveyed uses, or is designed to use, a multistage interconnection network. Together they illustrate a range of architectural ideas and a range of intended

application domains. Thus, this material provides a context in which to view the interconnection network and image processing studies of this research.

The Extra Stage Cube is a new fault-tolerant multistage interconnection network intended for use in large-scale SIMD, MSIMD, MIMD, and partitionable SIMD/MIMD computer systems, and is the network to be used in the PASM prototype under construction at Purdue University. Chapter 3 defines the Generalized Cube network as shows how the ESC is derived from it. The Generalized Cube was noted to be representative of a class of cube-type networks, including the STARAN flip, the omega, the indirect binary n-cube, and the SW-banyan ($S=F=2, L=n$) networks.

A formal development of the properties of the ESC was set forth in Chapter 4. The basic concepts of network fault model and fault-tolerance criterion were defined, and their meaning for the ESC chosen to reflect the anticipated operating environment of an interconnection network, rather than for ease of analysis. The ESC was proved to be single-fault tolerant with respect to its fault model and fault-tolerance criterion. A simple method was given for finding a fault-free path through a faulted network. The ESC was shown to tolerate some instances of multiple faults. Routing tags were developed that allow full access to the fault-tolerance capabilities of the ESC. Its partitioning and permuting were capabilities were described.

Because the field of fault-tolerant multistage interconnection networks for parallel/distributed computers is a new one, and because the ESC was described in the early literature, the survey of such networks was presented in Chapter 5, following the chapter describing the ESC. The challenging design goals chosen for and met by the ESC result in its retaining its merit as new networks have been proposed. The ESC was compared with the surveyed

fault-tolerant networks. The ESC is either less physically complex and equally fault-tolerant or more fault-tolerant than other fault-tolerant multistage interconnection networks that have been reported in the literature.

Chapter 6 presented a study of the terminal-pair reliability of the ESC. Reliability was measured as the probability that there exists at least one path between any network input and output. This extended the analytical work of Chapter 4 establishing single-fault tolerance. An exact solution for the case of two faults is developed based on the ESC fault model stated in Chapter 4.

From the reliability study of the ESC, opportunities for improved fault-tolerance at minimal additional hardware or operational protocol overhead cost were recognized. An Enhanced ESC was described and its capabilities studied in Chapter 7. Large networks were found to benefit more than small ones from this modification.

Alternative structures to traditional interchange boxes, motivated by VLSI considerations, were presented in Chapter 8. The performance of a 4×4 crossbar node and a composite node formed from four 2×2 crossbar elements was studied. Implementation of an ESC, or ESC-like, network using 4×4 switches was discussed.

Chapter 9 considered digital image contour extraction, an image processing task within the application domain for PASM and with wide-spread use. The value of the scenario approach was that it provided insight not available from an individual consideration of the algorithms comprising the contour extraction task. Numerous system design guidelines were uncovered for the operating system, processing element design, and interconnection network requirements. In addition, the parallel formulation of the task

provided the opportunity for faster processing speeds, elimination of object size constraints, and improved accuracy.

In summary, the contributions of this work are threefold. The first is the development and study of the Extra Stage Cube fault-tolerant, multistage interconnection network. The second is the analysis of the performance of alternative switching element structures for interconnection networks. The third is the examination in detail of an image processing scenario for PASM and the ESC network.

LIST OF REFERENCES

LIST OF REFERENCES

- [Ack77] D. L. Ackerman, "Algorithm design for digital image correlation on a parallel processing system," in *High Speed Computer and Algorithm Organization*, edited by D. J. Kuck, D. H. Lawrie, and A. H. Sameh, Academic Press, Inc., New York, 1977, pp. 307-308.
- [AdD84] G. B. Adams III and P. J. Denning, *A Ten Year Plan for Concurrent Processing Research*, Research Institute for Advanced Computer Science, TR 84.1, Mar. 1984, 31 pp.
- [AdS80] G. B. Adams III and H. J. Siegel, *Properties of the Augmented Data Manipulator in an SIMD Environment*, School of Electrical Engineering, Purdue University, TR-EE 80-51, Dec. 1980, 105 pp.
- [AdS82a] G. B. Adams III and H. J. Siegel, "A multistage network with an additional stage for fault tolerance," *15th Hawaii International Conference on System Sciences*, Jan. 1982, vol. 1, pp. 333-342.
- [AdS82b] G. B. Adams III and H. J. Siegel, "On the number of permutations performable by the augmented data manipulator network," *IEEE Transactions on Computers*, vol. C-31, pp. 270-277, Apr. 1982.
- [AdS82c] G. B. Adams III and H. J. Siegel, "Properties of the extra stage cube under multiple faults," *14th Southeastern Symposium on System Theory*, Apr. 1982, pp. 3-6.
- [AdS82d] G. B. Adams III and H. J. Siegel, "The extra stage cube: A fault-tolerant interconnection network for supersystems." *IEEE Transactions on Computers*, vol. C-31, pp. 443-454, May 1982. Also in *Tutorial: Interconnection Networks for Parallel and Distributed Processing*, edited by C. L. Wu and T. Y. Feng, IEEE Computer Society Press, Silver Spring, MD, 1984, pp. 397-408.

- [AdS82e] G. B. Adams III and H. J. Siegel, *Extra Stage Cube*, Patent Application, Serial Number 6-452438, Dec. 23, 1982.
- [AdS84a] G. B. Adams III and H. J. Siegel, "A survey of fault-tolerant multistage networks and comparison to the extra stage cube," *17th Hawaii International Conference on System Sciences*, Jan. 1984, vol. 1, pp. 268-277.
- [AdS84b] G. B. Adams III and H. J. Siegel, "The use of 4×4 switching elements in the multistage cube network," *1st International Conference on Computers and Applications*, June 1984, pp. 585-592.
- [AdS84c] G. B. Adams III and H. J. Siegel, "Modifications to improve the fault tolerance of the extra stage cube interconnection network," *1984 International Conference on Parallel Processing*, Aug. 1984, pp. 169-173.
- [AgK83] D. P. Agrawal and D. Kaur, "Fault tolerant capabilities of redundant multistage interconnection networks," *1983 Real-Time Systems Symposium*, Dec. 1983, pp. 119-127.
- [AgL84] D. P. Agrawal and J. S. Leu, "Dynamic accessibility testing and path length optimization of multistage interconnection networks," *4th International Conference on Distributed Computing Systems*, May 1984, pp. 266-277.
- [Agr82] D. P. Agrawal, "Testing and fault tolerance of multistage interconnection networks," *Computer*, vol. 15, pp. 41-53, Apr. 1982.
- [Agr83] D. P. Agrawal, "Graph theoretical analysis and design of multistage interconnection networks," *IEEE Transactions on Computers*, vol. C-32, pp. 637-648, July 1983.
- [AHU74] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*. Reading, Massachusetts: Addison-Wesley Publishing Company, 1974.

- [ANS78] ANSI X3.9-1978. *American National Standard Programming Language FORTRAN (Fortran 77)*. 1978.
- [ArP76] R. G. Arnold and E. W. Page, "A hierarchical restructurable multimicroprocessor architecture," *3rd Annual Symposium on Computer Architecture*, Jan. 1976, pp. 40-45.
- [Bae80] J. L. Baer, *Computer Systems Architecture*. Rockville, Maryland: Computer Science Press, 1980.
- [BaL81] G. H. Barnes and S. F. Lundstrum, "Design and validation of a connection network for many-processor multiprocessor systems," *Computer*, vol. 14, pp.31-41, Dec. 1981.
- [Bat74] K. E. Batcher, "STARAN parallel processor system hardware," *AFIPS 1974 National Computer Conference*, May 1974, pp. 405-410.
- [Bat76] K. E. Batcher, "The flip network in STARAN," *1976 International Conference on Parallel Processing*, Aug. 1976, pp. 65-71.
- [Bat77a] K. E. Batcher, "The multidimensional access memory in STARAN," *IEEE Transactions on Computers*, vol. C-26, pp. 174-177, Feb. 1977.
- [Bat77b] K. E. Batcher, "STARAN series E," *1977 International Conference on Parallel Processing*, Aug. 1977, pp. 140-143.
- [Bat80] K. E. Batcher, "Design of a massively parallel processor," *IEEE Transactions on Computers*, vol. C-29, pp. 836-840, Sept. 1980.
- [BBK68] G. H. Barnes, R. M. Brown, M. Kato, D. J. Kuck, D. L. Slotnick, and R. A. Stokes, "The Illiac IV computer," *IEEE Transactions on Computers*, vol. C-17, pp. 746-757, Aug. 1968.
- [BDM72] W. J. Bouknight, S. A. Denenberg, D. E. McIntyre, J. M. Randall, A. H. Sameh, and D. I. Slotnick, "The Illiac IV system," *Proceedings of the IEEE*, vol. 60, pp. 369-388, Apr. 1972.

- [Ben65] V. E. Beneš. *Mathematical Theory of Connecting Networks and Telephone Traffic*. New York: Academic Press, 1965.
- [BFH82] F. A. Briggs, K. S. Fu, K. Hwang, and B. W. Wah, "PUMPS architecture for pattern analysis and image database management," *IEEE Transactions on Computers*, vol. C-31, pp. 969-983, Oct. 1982.
- [BlB82] A. J. Blodgett and D. R. Barbour, "Thermal conduction module: A high-performance multilayer ceramic package," *IBM Journal of Research and Development*, vol. 26, pp. 30-36, Jan. 1982.
- [Bur79] Burroughs Corporation, *Final Report: Numerical Aerodynamic Simulation Facility Feasibility Study*, Burroughs Corporation Report, Contract Number NAS2-9897, Mar. 1979.
- [ChH82] R. T. Chin and C. A. Harlow, "Automated visual inspection: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-4, pp. 557-573, Nov. 1982.
- [ChM82] S. Cheemalavagu and M. Malek, "Analysis and simulation in banyan interconnection networks with 2×2 , 4×4 and 8×8 switching elements," *1982 Real-Time Systems Symposium*, Dec. 1982, pp. 83-89.
- [CiS80] L. Ciminiera and A. Serra, "LSI implementation of modular interconnection networks for MIMD machines," *1980 International Conference on Parallel Processing*, Aug. 1980, pp. 161-162.
- [CiS81] L. Ciminiera and A. Serra, "Modular interconnection networks with asynchronous control," *14th Hawaii International Conference on System Sciences*, Jan. 1981, vol. 1, pp. 210-218.
- [CiS82] L. Ciminiera and A. Serra, "A fault-tolerant connecting network for multiprocessor systems," *1982 International Conference on Parallel Processing*, Aug. 1982, pp. 113-122.

- [Clo53] C. Clos, "A study of non-blocking switching networks," *Bell System Technical Journal*, vol. 32, pp. 406-424, Mar. 1953.
- [CIS83] C. Cline and H. J. Siegel, "Extensions of Ada for SIMD parallel processing," *7th International Computer Software and Applications Conference (COMPSAC '83)*, Nov. 1983, pp. 366-372.
- [daC78] A. L. daCosta, "Application of computational aerodynamics methods to the design and analysis of transport aircraft," ICAS Paper No. 78.B2-01, Sept. 1978.
- [DBL80] J. B. Dennis, G. A. Boughton, and C. K. C. Leung, "Building blocks for data flow prototypes," *7th Annual Symposium on Computer Architecture*, May 1980, pp. 1-8.
- [Den70] P. J. Denning, "Virtual memory," *Computing Surveys*, vol. 2, pp. 153-188, Sept. 1970.
- [Dia81] D. M. Dias, *Packet Communication in Delta and Related Networks*, Ph.D. Dissertation, Department of Electrical Engineering, Rice University, Apr. 1981.
- [DiJ80] D. M. Dias and J. R. Jump, "Packet communication in multistage shuffle-exchange networks," *1980 International Conference on Parallel Processing*, Aug. 1980, pp. 327-328.
- [DiJ81] D. M. Dias and J. R. Jump, "Analysis and simulation of buffered delta networks," *IEEE Transactions on Computers*, vol. C-30, pp. 273-282, Apr. 1981.
- [DiJ82] D. M. Dias and J. R. Jump, "Augmented and pruned $N \log N$ multistage networks: Topology and performance," *1982 International Conference on Parallel Processing*, Aug. 1982, pp. 10-11.
- [Dij68] E. W. Dijkstra, "Cooperating sequential processes," in *Programming Languages*, edited by F. Genuys, Academic Press, Inc., New York, 1968, pp. 43-112.

- [Dou82] R. J. Douglass, "A pipeline architecture for image segmentation," *15th Hawaii International Conference on System Sciences*, Jan. 1982, vol. 1, pp. 360-367.
- [DuH73] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*. New York: Wiley, 1973.
- [FeK82] T. Feng and I. Kao, "On fault-diagnosis of some multistage networks," *1982 International Conference on Parallel Processing*, Aug. 1982, pp. 99-101.
- [Fen74] T. Y. Feng, "Data manipulating functions in parallel processors and their implementations," *IEEE Transactions on Computers*, vol. C-23, pp. 309-318, Mar. 1974.
- [FeW81] T. Y. Feng and C. L. Wu, "Fault-diagnosis for a class of multistage interconnection networks," *IEEE Transactions on Computers*, vol. C-30, pp. 743-758, Oct. 1981.
- [Fly66] M. J. Flynn, "Very high-speed computing systems," *Proceedings of the IEEE*, vol. 54, pp. 1901-1909, Dec. 1966.
- [Fos76] C. C. Foster, *Content Addressable Parallel Processors*. New York: Van Nostrand Reinhold, 1976.
- [Fra81] M. A. Franklin, "VLSI performance comparison of banyan and crossbar communications networks," *IEEE Transactions on Computers*, vol. C-30, pp. 283-291, Apr. 1981.
- [GaW81] N. C. Gallagher, Jr. and G. L. Wise, "Passband and stepband properties of median filters," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. ASSP-29, pp. 1136-1141, Dec. 1981.
- [GGK83] 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 Transactions on Computers*, vol. C-32, pp. 175-189, Feb. 1983.

- [GoL73] L. R. Goke and G. J. Lipovski, "Banyan networks for partitioning multimicroprocessor systems," *1st Symposium on Computer Architecture*, Dec. 1973, pp. 21-28.
- [Hu62] M. K. Hu, "Visual pattern recognition by moment invariants," *IRE Transactions on Information Theory*, vol. IT-8, pp. 179-187, Feb. 1962.
- [HuM81] A. C. Hung and M. Malek, "A 4x4 modular crossbar design for the multistage interconnection networks," *1981 Real-Time Systems Symposium*, Dec. 1981, pp. 3-12.
- [Jar80] J. F. Jarvis, "A method for automating the visual inspection of printed wiring boards," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-2, pp. 77-82, Jan. 1980.
- [JoJ72] D. E. Johnson and J. R. Johnson, *Graph Theory with Engineering Applications*. New York: Ronald Press, 1972.
- [KaK79] S. I. Kartashev and S. P. Kartashev, "A multicomputer system with dynamic architecture," *IEEE Transactions on Computers*, vol. C-28, pp. 704-720, Oct. 1979.
- [Knu76] D. E. Knuth, "Big omicron, big omega, and big theta," *SIGACT News*, vol. 8, no. 2, pp. 18-24.
- [KPL80] R. N. Kapur, U. V. Premkumar, and G. J. Lipovski, "Organization of the TRAC processor-memory subsystem," *AFIPS 1980 National Computer Conference*, June 1980, pp. 623-629.
- [KrM83] C. P. Kruskal and M. Snir, "The performance of multistage interconnection networks for multiprocessors," *IEEE Transactions on Computers*, vol. C-32, pp. 1091-1098, Dec. 1983.
- [KSG83] J. T. Kuehn, H. J. Siegel, and M. J. Grosz, "A distributed memory management system for PASM," *IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Data Base Management*, Oct. 1983, pp. 101-108.

- [KSH82] J. T. Kuehn, H. J. Siegel, and P. D. Hallenbeck, "Design and simulation of an MC68000-based multimicroprocessor system," *1982 International Conference on Parallel Processing*, Aug. 1982, pp. 353-362.
- [KST85] 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* edited by S. Levialdi, Academic Press, to appear 1985.
- [Law75] D. H. Lawrie, "Access and alignment of data in an array processor," *IEEE Transactions on Computers*, vol. C-24, pp. 1145-1155, Dec. 1975.
- [Lee84] K. Y. Lee, "A new Beneš control algorithm," *1984 International Conference on Parallel Processing*, Aug. 1984, pp. 51-58.
- [Len78] J. Lenfant, "Parallel permutations of data: A Beneš network control algorithm for frequently used permutations," *IEEE Transactions on Computers*, vol. C-27, pp. 637-647, July 1978.
- [LeS83] D. C. H. Lee and J. P. Shen, "Easily-testable (N,K) shuffle/exchange networks," *1983 International Conference on Parallel Processing*, Aug. 1983, pp. 65-70.
- [Lim82] W. Y. Lim, "A test strategy for packet switching networks," *1982 International Conference on Parallel Processing*, Aug. 1982, pp. 96-98.
- [Lin70] S. Lin, *An Introduction to Error Correcting Codes*. Englewood Cliffs, NJ: Prentice-Hall, 1970, p. 43.
- [LiT77] G. J. Lipovski and A. Tripathi, "A reconfigurable varistructure array processor," *1977 International Conference on Parallel Processing*, Aug. 1977, pp. 165-174.

- [LLY82] J. E. Lilienkamp, D. H. Lawrie, and P. C. Yew, "A fault tolerant interconnection network using error correcting codes," *1982 International Conference on Parallel Processing*, Aug. 1982, pp. 123-125.
- [MAS81] R. J. McMillen, G. B. Adams III, and H. J. Siegel, "Performance and implementation of 4x4 switching nodes in an interconnection network for PASM," *1981 International Conference on Parallel Processing*, Aug. 1981, pp. 229-233.
- [McM82] R. J. McMillen, *A Study of Multistage Interconnection Networks: Design, Distributed Control, Fault Tolerance, and Performance*, Ph.D. Dissertation, School of Electrical Engineering, Purdue University, Dec. 1982.
- [McS80] R. J. McMillen and H. J. Siegel, "The hybrid cube network," *Distributed Data Acquisition, Computing, and Control Symposium*, Dec. 1980, pp. 11-22.
- [McS82a] R. J. McMillen and H. J. Siegel, "Performance and fault tolerance improvements in the inverse augmented data manipulator network," *9th Annual Symposium on Computer Architecture*, Apr. 1982, pp. 63-72.
- [McS82b] R. J. McMillen and H. J. Siegel, "A comparison of cube type and data manipulator type networks," *3rd International Conference on Distributed Computing Systems*, Oct. 1982, pp. 614-621.
- [McS82c] R. J. McMillen and H. J. Siegel, "Routing schemes for the augmented data manipulator network in an MIMD system," *IEEE Transactions on Computers*, vol. C-31, pp. 1202-1214, Dec. 1982.
- [McW78] W. C. McDonald and J. M. Williams, "The advanced data processing testbed," *2nd International Computer Software and Applications Conference (COMPSAC '78)*, Mar. 1978, pp. 346-351.

- [MDS82] T. N. Mudge, E. J. Delp, L. J. Siegel, and H. J. Siegel, "Image coding using the multimicroprocessor system PASM," *1982 IEEE Computer Society Conference on Pattern Recognition and Image Processing*, June 1982, pp. 200-205.
- [Mey70] P. L. Meyer, *Introductory Probability and Statistical Analysis - Second Edition*. Reading, Massachusetts: Addison-Wesley, 1970, pp. 225-239.
- [MiD83] O. R. Mitchell and K. A. Dunkelbereger, private communication, 1983.
- [Mil79] D. L. Milgram, "Region extraction using convergent evidence," *Computer Graphics and Image Processing*, vol. 11, pp. 1-12, Sept. 1979.
- [MRF81] O. R. Mitchell, A. P. Reeves, and K. S. Fu, "Shape and texture measurements for automated cartography," *1981 IEEE Computer Society Conference on Pattern Recognition and Image Processing*, Aug. 1981, p. 367.
- [MSS80a] P. T. Mueller, Jr., L. J. Siegel, and H. J. Siegel, "A parallel language for image and speech processing," *4th International Computer Software and Applications Conference (COMPSAC '80)*, Oct. 1980, pp. 476-483.
- [MSS80b] P. T. Mueller, Jr., L. J. Siegel, and H. J. Siegel, "Parallel algorithms for the two-dimensional FFT," *5th International Conference on Pattern Recognition*, Dec. 1980, pp. 497-502.
- [NeP82] R. Nevitia and K. E. Price, "Locating structures in aerial images," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-4, pp. 476-484, Sept. 1982.
- [Nut77] G. J. Nutt, "Microprocessor implementation of a parallel processor," *4th Annual Symposium on Computer Architecture*, Mar. 1977, pp. 147-152.

- [OpT71] D. C. Opferman and N. T. Tsao-Wu, "On a class of rearrangeable switching networks - Part I: Control algorithm," *Bell System Technical Journal*, vol. 50, pp. 1601-1618, May-June 1971.
- [Pad84] K. Padmanabhan, *Fault Tolerance and Performance Improvement in Multiprocessor Interconnection Networks*, Ph.D. Dissertation, University of Illinois at Champaign-Urbana, 1984.
- [PaL83a] K. Padmanabhan and D. H. Lawrie, "Fault tolerance schemes in shuffle-exchange type interconnection networks," *1983 International Conference on Parallel Processing*, Aug. 1983, pp. 71-75.
- [PaL83b] K. Padmanabhan and D. H. Lawrie, "A class of redundant path multistage interconnection networks," *IEEE Transactions on Computers*, vol. C-32, pp. 1099-1108, Dec. 1983.
- [Pap65] A. Papoulis, *Probability, Random Variables, and Stochastic Processes*. New York: McGraw-Hill, 1965.
- [PaR82] D. S. Parker and C. S. Raghavendra, "The gamma network: A multiprocessor interconnection network with redundant paths," *9th Annual Symposium on Computer Architecture*, Apr. 1982, pp. 73-80.
- [PaR84] D. S. Parker and C. S. Raghavendra, "The gamma network," *IEEE Transactions on Computers*, vol. C-33, pp. 367-373, Apr. 1984.
- [Pat81] J. H. Patel, "Performance of processor-memory interconnections for multiprocessors," *IEEE Transactions on Computers*, vol. C-30, pp. 771-780, Oct. 1981.
- [Pea77] M. C. Pease, III, "The indirect binary n-cube microprocessor array," *IEEE Transactions on Computers*, vol. C-26, pp. 458-473, May 1977.
- [PKM80] U. V. Premkumar, R. Kapur, M. Malek, G. J. Lipovski, and P. Horne, "Design and implementation of the banyan interconnection network in TRAC," *AFIPS 1980 National Computer Conference*, June 1980, pp. 643-653.

- [PrM66] J. M. S. Prewitt and M. L. Mendelsohn, "The analysis of cell images," *Annals N.Y. Academy of Science*, vol. 128, pp. 1035-1053, 1966.
- [RaP84] C. S. Raghavendra and D. S. Parker, "Reliability analysis of an interconnection network," *4rd International Conference on Distributed Computing Systems*, May 1984, pp. 461-471.
- [ReK84] S. M. Reddy and V. P. Kumar, "On Fault-Tolerant Multistage Interconnection Networks," *1984 International Conference on Parallel Processing*, Aug. 1984, pp. 155-164.
- [RiA77] E. M. Riseman and M. A. Arbib, "Computational techniques in the visual segmentation of static scenes," *Computer Graphics and Image Processing*, vol. 6, pp. 221-276, 1977.
- [RoK82] A. Rosenfeld and A. C. Kak, *Digital Picture Processing (Second Edition, Volume I)*. New York: Academic Press, 1982.
- [RuT83] P. E. Rubbert and E. N. Tinoco, "Impact of computational methods in aircraft design," *AIAA Atmospheric Flight Mechanics Conference*, Aug. 1983, AIAA-83-2060.
- [SFS77] R. J. Swan, S. H. Fuller, and D. P. Siewiorek, "Cm*: A modular multi-microprocessor," *AFIPS 1977 National Computer Conference*, June 1977, pp. 637-644.
- [She82] J. P. Shen, "Fault tolerance analysis of several interconnection networks," *1982 International Conference on Parallel Processing*, Aug. 1982, pp. 102-112.
- [ShH80] J. P. Shen and J. P. Hayes, "Fault tolerance of a class of connecting networks," *7th Annual Symposium on Computer Architecture*, May 1980, pp. 61-71.
- [ShH84] J. P. Shen and J. P. Hayes, "Fault-tolerance of dynamic-full-access interconnection networks," *IEEE Transactions on Computers*, vol. C-33, pp. 241-248, Mar. 1984.

- [Sie77a] H. J. Siegel, "Analysis techniques for SIMD machine interconnection networks and the effects of processor address masks," *IEEE Transactions on Computers*, vol. C-26, pp. 153-161, Feb. 1977.
- [Sie77b] H. J. Siegel, "Controlling the active/inactive status of SIMD machine processors," *1977 International Conference on Parallel Processing*, Aug. 1977, p. 183.
- [Sie79a] H. J. Siegel, "Interconnection networks for SIMD machines," *Computer*, vol. 12, pp. 57-65, June 1979.
- [Sie79b] H. J. Siegel, "A model of SIMD machines and a comparison of various interconnection networks," *IEEE Transactions on Computers*, vol. C-28, pp. 907-917, Dec. 1979.
- [Sie80] H. J. Siegel, "The theory underlying the partitioning of permutation networks," *IEEE Transactions on Computers*, vol. C-29, pp. 791-801, Sept. 1980.
- [Sie85] H. J. Siegel, *Interconnection Networks for Large Scale Parallel Processing: Theory and Case Studies*. Lexington, Massachusetts: Lexington Books, to appear 1985.
- [SiM81a] H. J. Siegel and R. J. McMillen, "Using the augmented data manipulator network in PASM," *Computer*, vol. 14, pp. 25-33, Feb. 1981.
- [SiM81b] H. J. Siegel and R. J. McMillen, "The multistage cube: A versatile interconnection network," *Computer*, vol. 14, pp. 65-76, Dec. 1981.
- [SiS78] H. J. Siegel and S. D. Smith, "Study of multistage SIMD interconnection networks," *5th Annual Symposium on Computer Architecture*, Apr. 1978, pp. 223-229.
- [SKW79] H. J. Siegel, F. Kemmerer, and M. Washburn, "Parallel memory system for a partitionable SIMD/MIMD machine," *1979 International Conference on Parallel Processing*, Aug. 1979, pp. 212-221.

- [Smi81] S. D. Smith, "LSI design considerations for multistage interconnection networks for parallel processing systems," *14th Hawaii International Conference on System Sciences*, Jan. 1981, vol. 1, pp. 219-227.
- [SMM79] H. J. Siegel, R. J. McMillen, and P. T. Mueller, Jr., "A survey of interconnection methods for reconfigurable parallel processing systems," *AFIPS 1979 National Computer Conference*, June 1979, pp. 529-542.
- [SMS78] H. J. Siegel, P. T. Mueller, Jr., and H. E. Smalley, Jr., "Control of a partitionable multimicroprocessor system," *1978 International Conference on Parallel Processing*, Aug. 1978, pp. 9-17.
- [SoR80] S. Sowrirajan and S. M. Reddy, "A design for fault-tolerant full connection networks," *1980 Conference on Information Sciences and Systems*, Mar. 1980, pp. 536-540.
- [SSD84] H. J. Siegel, T. Schwederski, N. J. Davis IV, J. T. Kuehn, "PASM: A reconfigurable parallel system for image processing," *Workshop on Algorithm-Guided Parallel Architectures for Automatic Target Recognition*, July 1984, proceedings to appear; reprinted in *ACM SIGARCH Computer Architecture News*, vol. 12, pp. 7-19, Sept. 1984.
- [SSE80] P. H. Swain, H. J. Siegel, and J. El-Achkar, "Multiprocessor implementation of pattern recognition: A general approach," *5th International Conference on Pattern Recognition*, Dec. 1980, pp. 309-317.
- [SSF82] L. J. Siegel, H. J. Siegel, and A. E. Feather, "Parallel processing approaches to image correlation," *IEEE Transactions on Computers*, vol. C-31, pp. 208-218, Mar. 1982.
- [SSK81] H. J. Siegel, L. J. Siegel, F. C. 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 Transactions on Computers*, vol. C-30, pp. 934-947, Dec. 1981.

- [SSM80] S. D. Smith, H. J. Siegel, R. J. McMillen, and G. B. Adams III, "Use of the augmented data manipulator multistage network for SIMD machines," *1980 International Conference on Parallel Processing*, Aug. 1980, pp. 75-78.
- [SSS82] L. J. Siegel, H. J. Siegel, and P. H. Swain, "Performance measures for evaluating algorithms for SIMD machines," *IEEE Transactions on Software Engineering*, vol. SE-8, pp. 319-331, July 1982.
- [Sta74] C. D. Stamopoulos, "Parallel algorithms for joining two points by a straight line segment," *IEEE Transactions on Computers*, vol. C-23, pp. 642-646, June 1974.
- [Sto80] H. S. Stone, "Parallel computers," in *Introduction to Computer Architecture*, 2nd edition, edited by H. S. Stone, Chicago: Science Research Associates, Inc., 1980, pp. 363-425.
- [SUK80] 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," *AFIPS 1980 National Computer Conference*, June 1980, pp. 631-641.
- [SuR82] R. E. Suci and A. P. Reeves, "A comparison of differential and moment based edge detectors," *1982 IEEE Computer Society Conference on Pattern Recognition and Image Processing*, June 1982, pp. 97-102.
- [TAS83] D. L. Tuomenoksa, G. B. Adams III, H. J. Siegel, and O. R. Mitchell, "A parallel algorithm for contour extraction: Advantages and architectural implications," *Computer Vision and Pattern Recognition 1983 Symposium*, June 1983, pp. 336-344.
- [Tea80] M. R. Teague, "Image analysis via the general theory of moments," *Journal of the Optical Society of America*, vol. 70, pp. 920-933, Aug. 1980.
- [ThN83] S. Thanawastien and V. P. Nelson, "Distributed path testing in a shuffle/exchange network based on a write/verify approach," *1983 Real-Time Systems Symposium*, Dec. 1983, pp. 131-140.

- [Thu74] K. J. Thurber, "Interconnection networks - a survey and assessment," *AFIPS 1974 National Computer Conference*, May 1974, pp. 909-919.
- [TiC84] E. N. Tinoco and A. W. Chen, "Transonic CFD applications to engine/airframe integration," *AIAA 22nd Aerospace Sciences Meeting*, Jan. 1984, AIAA-84-0381.
- [Tin84] E. N. Tinoco, private communication, 1984.
- [TrL79] A. R. Tripathi and G. J. Lipovski, "Packet switching banyan networks," *6th Annual Symposium on Computer Architecture*, Apr. 1979, pp. 160-167.
- [Tuo83] D. L. Tuomenoksa, *Design of the Operating System for the PASM Parallel Processing System*, Ph.D. Dissertation, School of Electrical Engineering, Purdue University, May 1983.
- [TuS82a] D. L. Tuomenoksa and H. J. Siegel, "Analysis of the PASM control system memory hierarchy," *1982 International Conference on Parallel Processing*, Aug. 1982, pp. 363-370.
- [TuS82b] D. L. Tuomenoksa and H. J. Siegel, "Analysis of multiple-queue task scheduling algorithms for multiple-SIMD machines," *3rd International Conference on Distributed Computing Systems*, Oct. 1982, pp. 114-121.
- [TuS83] D. L. Tuomenoksa and H. J. Siegel, "Preloading schemes for the PASM parallel memory system," *1983 International Conference on Parallel Processing*, Aug. 1983, pp. 407-415.
- [TuS84a] D. L. Tuomenoksa and H. J. Siegel, "A distributed operating system for PASM," *17th Hawaii International Conference on System Sciences*, Jan. 1984, vol. 1, pp. 69-77.
- [TuS84b] D. L. Tuomenoksa and H. J. Siegel, "Task preloading schemes for reconfigurable parallel processing systems," *IEEE Transactions on Computers*, vol. C-33, pp. 895-905, Oct. 1984.

- [WaS82] M. R. Warpenburg and L. J. Siegel, "Image resampling in an SIMD environment," *IEEE Transactions on Computers*, vol. C-31, pp. 934-942, Oct. 1982.
- [WaW80] T. P. Wallace and P. A. Wintz, "An efficient three-dimensional aircraft recognition algorithm using normalized Fourier descriptors," *Computer Graphics and Image Processing*, vol. 13, pp. 99-126, June 1980.
- [Wen76] K. Y. Wen, *Interprocessor Connections - Capabilities, Exploitation, and Effectiveness*, Ph.D. Dissertation, University of Illinois at Champagne-Urbana, Report UIUCDCS-R-76-830, Oct. 1976, 170 pp.
- [WFL82] C. L. Wu, T. Y. Feng, and M. C. Lin, "Star: A local network system for real-time management of imagery data," *IEEE Transactions on Computers*, vol. C-31, pp. 923-933, Oct. 1982.
- [WuB72] W. A. Wulf and C. G. Bell, "C.mmp - a multi-miniprocessor," *Fall Joint Computer Conference*, Dec. 1972, pp. 765-777.
- [WuF80] C. L. Wu and T. Y. Feng, "On a class of multistage interconnection networks," *IEEE Transactions on Computers*, vol. C-29, pp. 694-702, Aug. 1980.

APPENDICES

Appendix A

**Program to Count Lossy Pairs
with Stage Bypassing**

```

/*
** Program to directly enumerate fault pairs and lossy pairs for the ZSC
** with STAGE bypassing.
** Used to confirm the equations for counting these objects.
** Run giving the value of n as the argument.
**
**
** WARNING >> For n = 6 run time is 123.9 CPU HOURS <<
**             For n = 5 run time is slightly over 4.3 hours
**             For n = 4 run time is slightly over 11 minutes
**             For n = 3 run time is 18 seconds
**             For n = 2 run time is less than 1 second
**
** Options: -v verbose      lists information on each fault pair (disabled);
**                               used for debugging
**
** Variables:  n            number of network stages
**                               (not counting extra stage)
**             N            number of network inputs
**             bbfault      number of EB-fault pairs
**             bblossy      number of EB-lossy pairs
**             lbfault      number of LB-fault pairs
**             lblossy      number of LB-lossy pairs
**             llfault      number of LL-fault pairs
**             lllossy      number of LL-lossy pairs
**             mask          constant array containing bit masks used to
**                               compute the stage outputs of the primary and
**                               secondary paths given the s/d pair and stage
*/

#include <stdio.h>

int bads, loss, verbose;

typedef unsigned long uint;
uint n, N;

struct {
    uint stage;
    uint link;
} bad[4];

main(argc, argv)
int argc;
char *argv[];
{
    register uint a,b;
    register int i,j;
    int bblossy, lblossy, lllossy, bbfault, lbfault, llfault;

```

```

int oldloss;
verbose = 0;
if ( argc==3 ) {
    if ( strcmp(argv[1],"-v")==0)
        verbose = 1;
    else {
        fprintf(stderr,"usage: %s [-v] n\n",argv[0]);
        exit(1);
    }
    argc--;
    argv++;
}
if ( argc!=2 ) {
    fprintf(stderr,"usage: %s n\n",argv[0]);
    exit(1);
}
n = atoi(argv[1]);
N = 1<<n;
bblossy = llossy = lllossy = bbfault = lbfault = llfault = 0;

/* BB-fault pair section */
loss = 0;
for ( i=n ; i>=0 ; i-- ) {
/* for all stages */
    for(a=0 ; a<N/2 ; a++ ) {
/* for the "upper" output of each box in stage i */
        for ( j=i ; j>=0 ; j-- ) {
/* for all stages less than or equal to i */
            for ( b=0 ; b<N/2 ; b++ ) {
/* for the "upper" outputs of each box in stage j */
                if ( i==j && b<=a )
                    continue;
/* unless i=j, then for the "upper" output of each stage j box with */
/* outputs greater than a */
                    bbfault = bbfault + 1;
                    if ( i==j && (i==n | i==0) )
                        continue;
/* with both faults confined to stage n or stage 0 the ESC continues to */
/* function; there is no need to test such fault pairs for loss of FFIC */
                    if ( (i==n && j!=n) | (i==0 && j==0) ) {
                        bblossy++;
                        continue;
                    }
/* a stage n or stage 0 box fault combined with a box fault outside of that */
/* stage is lossy; count it as such and go on to the next fault pair */
                    oldloss = loss;
                    bad[0].stage = i;
                    bad[0].link = zeroins(i,a,n);

```

```

bad[1].stage = i;
bad[1].link = lower(i,zeroins(i,a,n),n);
bad[2].stage = j;
bad[2].link = zeroins(j,b,n);
bad[3].stage = j;
bad[3].link = lower(j,zeroins(j,b,n),n);
bads = 4;

/*
if (verbose)
    printf("### (%2d,%2d) & (%2d,%2d)\n
& (%2d,%2d) & (%2d,%2d)\n",i,bad[0].link,i,bad[1].link,j,bad[2].link,j,bad[3].link);
*/

test();
if (loss>oldloss)
    bblossy++;
    }
    }
}

printf("Number EB-lossy pairs = %d\n", bblossy);
printf("Number EB-fault pairs = %d\n", bbfault);

/* LB-fault pair section */
loss = 0;
for ( i=n ; i>0 ; i-- ) {
/* for all stages greater than 0 (there are no LB-fault pairs with */
/* the "uppermost fault" in stage 0) */

    for(a=0 ; a<N/2 ; a++ ) {
/* assume the box fault is in stage i; for the "upper" output of each box */
        for ( j=i ; j>0 ; j-- ) {
/* for all stages less than or equal to i and */
/* greater than 0 (there are no stage 0 links) */
            for ( b=0 ; b<N; b++ ) {
/* for all outputs in stage j (which must contain the link fault) */
                lbfault++;
                if ( i==n ) {
                    lblossy++;
                    continue;
                }
            }
/* an LB-fault pair containing a stage n box fault is a lossy pair */
/* no need to test */

            oldloss = loss;
            bad[0].stage = i;
            bad[0].link = zeroins(i,a,n);
            bad[1].stage = i;
            bad[1].link = lower(i,zeroins(i,a,n),n);
            bad[2].stage = j;

```

```

        bad[2].link = b;
        bads = 3;
        if (verbose)
            printf("### (%2d,%2d) & (%2d,%2d)\n
& (%2d,%2d)\n", bad[0].stage, bad[0].link, bad[1].stage, bad[1].link, bad[2].stage, bad[2].link);
            test();
            if (loss > oldloss)
                lblossy++;
        }
    }

    for(a=0 ; a<N ; a++) {
/* assume now that the link fault is in stage i; for each stage i output */
        for ( j=i-1 ; j>=0 ; j-- ) {
/* for all stages less than i */
            for ( b=0 ; b<N/2 ; b++ ) {
/* for the "upper" outputs of each box in stage j */
                lbfault = lbfault + 1;
                if ( j==0 ) {
                    lblossy++;
                    continue;
                }
/* an LB-fault pair containing a stage 0 box fault is a lossy pair */
/* no need to test */

                oldloss = loss;
                bad[0].stage = i;
                bad[0].link = a;
                bad[1].stage = j;
                bad[1].link = zeroins(j,b,N);
                bad[2].stage = j;
                bad[2].link = lower(j,zeroins(j,b,N),N);
                bads = 3;
                if (verbose)
                    printf("### (%2d,%2d) & (%2d,%2d)\n
& (%2d,%2d)\n", bad[0].stage, bad[0].link, bad[1].stage, bad[1].link, bad[2].stage, bad[2].link);
                    test();
                    if (loss > oldloss)
                        lblossy++;
            }
        }
    }

    printf("Number LB-lossy pairs = %d\n", lblossy);
    printf("Number LB-fault pairs = %d\n", lbfault);

/* LL-fault pair section */
    loss = 0;

```



```

        for ( i=a ; i>0 ; i-- ) {
/* for all stages with links (i.e., excluding stage 0) */
            for(a=0 ; a<N ; a++) {
/* for all outputs of stage i */
                for ( j=i ; j>0 ; j-- ) {
/* for all stages less than or equal to i that have links */
                    for ( b=0 ; b<N ; b++ ) {
/* for all outputs of stage j */
                        if ( b<=a && i==j )
                            continue;
/* unless i=j, then for all stage j outputs greater than a */
                            llfault = llfault + 1;
/* count number of LL-faulty pairs */
                                oldloss = loss;
                                bad[0].stage = i;
                                bad[0].link = a;
                                bad[1].stage = j;
                                bad[1].link = b;
                                bads = 2;
/*
                                if (verbose)
                                    printf("### (%2d,%2d) & (%2d,%2d)\n\n
".i,a,j,b); */
                                    test();
                                    if (loss > oldloss)
                                        llossy++;
/* count number of LL-lossy pairs */
                                }
                            }
                        }
                    }
                }
            }
        }
        printf("Number LL-lossy pairs = %d\n", llossy);
        printf("Number LL-fault pairs = %d\n", llfault);
    }

uint mask[] = {0,0x1,0x3,0x7,0xf,0x1f,0x3f,0x7f,0xff,0x1ff,0x3ff,0x7ff,0xfff,
0x1fff,0x3fff,0x7fff,0xffff,0x1ffff,0x3ffff,0x7ffff,0xfffff,
0x1fffff,0x3fffff,0x7fffff,0xfffff,0x1fffff,0x3fffff,0x7fffff,
0xfffff,0x1fffff,0x3fffff,0x7fffff};

test()
/*
** Test paths through the network until a source/destination pair that
** cannot communicate is found, or all pairs have been tested.
** Assumes that the bad links have already been listed in bad[*].
**
** Variables:  s      source
**             d      destination
**             stage  the number of the stage to be used in computing stage

```

```

**          outputs for the s/d pair being tested
**          link   output of stage "stage" taken by a path of the s/d pair
**                being tested
**          fault1 flag indicating the s/d pair has a faulty primary path
**          fault2 flag indicating the s/d pair has a faulty secondary path
**
** Affects global variable loss.
*/
(
  register int stage, fault1, fault2;
  register uint s, d, link;
  for ( s=0 ; s<N ; s++ ) {
    for ( d=0 ; d<N ; d++ ) {
      fault1 = fault2 = 0;
      for ( stage=n ; stage>=0 ; stage-- ) {
        link = (s&mask[stage])|(d&(~mask[stage]));
/* compute stage output of primary path */
        if ( faulty(stage,link) ) {
          fault1++;
          break;
        }
      }
      for ( stage=n ; stage>0 ; stage-- ) {
        link = ((s&mask[stage])|(d&(~mask[stage]))) ^ 1;
/* compute stage output of secondary path */
        if ( faulty(stage,link) ) {
          fault2++;
          break;
        }
      }
      if ( verbose ) {
        printf("%2d to %2d ", s, d);
        printf(" primary: ");
        if(fault1>0) printf("faulty at stage %d.",stage);
        else printf("okay.");
        printf(" secondary: ");
        if(fault2>0) printf("faulty at stage %d.",stage);
        else printf("okay.");
        printf(" %s\n", (fault1>0 && fault2>0)?"LOSS":"NOLOSS");
      }
      if ( fault1>0 && fault2>0 )
        loss++;
    }
  }

  faulty(stage,link)

```

```

/*
** Check a stage output against the list of fault labels.
** If there is a match, return a 1; else return 0.
*/
uint stage, link;
{
    int i;
    for ( i=0 ; i<badc ; i++ )
        if ( stage==bad[i].stage && link==bad[i].link )
            return 1;
    return 0;
}

lower(stage, link, n)
/*
** Generate the corresponding "lower" box output label
** given the stage number, label of the "upper" output,
** and the number of stages
*/
uint stage, link, n;
{
    if (stage == n)
        return(link ^ 1);
    else
        return(link ^ (1 << stage));
}

zeroin(stage, link, n)
/*
** Generate the "upper" box output label given the number of the
** stage containing the box, the "number" of the box within that
** stage (counting from zero with the "upper" box in the stage),
** and the number of stages
*/
uint stage, link, n;
{
    uint upper, lower;
    if (stage == n)
        stage = 0;
    upper = (link & (~0 << stage)) << 1;
    lower = link & (~0 << stage);
    return( upper | lower );
}

```

Appendix B

**Program to Count Lossy Pairs
with Box Bypassing**

Program to directly enumerate fault pairs and lossy pairs for the ESC
 with box bypassing.

to confirm the equations for counting these objects.
 giving the value of n as the argument.

PAING >> For n = 6 run time is 6.8 CPU DAYS <<
 For n = 5 run time is roughly 6 hours
 For n = 4 run time is slightly under 19 minutes
 For n = 3 run time is 29 seconds
 For n = 2 run time is less than 1 second

Options: -v verbose lists information on each fault pair (disabled);
 used for debugging

Variables: n number of network stages
 (not counting extra stage)
 N number of network inputs
 bbfault number of EB-fault pairs
 bblossy number of EB-lossy pairs
 lbfault number of LB-fault pairs
 lblossy number of LB-lossy pairs
 llfault number of LL-fault pairs
 lllossy number of LL-lossy pairs
 mask constant array containing bit masks used to
 compute the stage outputs of the primary and
 secondary paths given the s/d pair and stage

#include <stdio.h>

int loss, verbose;

typedef unsigned long uint;

uint

uint

uint stage;

uint link;

uint[4];

int argc, argv)

int argc;

int *argv[];

register uint a,b;

register int i,j;

int bblossy, lblossy, lllossy, bbfault, lbfault, llfault;

```

int oldloss;
verbose = 0;
if ( argc==3 ) {
    if ( strcmp(argv[1],"-v")==0)
        verbose = 1;
    else {
        fprintf(stderr,"usage: %s [-v] n\n",argv[0]);
        exit(1);
    }
    argc--;
    argv++;
}
if ( argc!=2 ) {
    fprintf(stderr,"usage: %s n\n",argv[0]);
    exit(1);
}
n = atoi(argv[1]);
N = 1<<n;
bblossy = lblossy = lllossy = bbfault = lbfault = llfault = 0;

/* EB-fault pair section */
loss = 0;
for ( i=n ; i>=0 ; i-- ) {
/* for all stages */
    for(a=0 ; a<N/2 ; a++ ) {
/* for the "upper" output of each box in stage i */
        for ( j=1 ; j>=0 ; j-- ) {
/* for all stages less than or equal to i */
            for ( b=0 ; b<N/2 ; b++ ) {
/* for the "upper" outputs of each box in stage j */
                if ( i==j && b<=a )
                    continue;
/* unless i=j, then for the "upper" output of each stage j box with */
/* outputs greater than a */
                    bbfault++;
                    if ( i==j && (i==n | i==0) )
                        continue;
/* with both faults confined to stage n or stage 0 the ESC continues to */
/* function; there is no need to test such fault pairs for loss of FFIC */
                    oldloss = loss;
                    if ( i!=j && i==n ) {
/* if there is one stage n fault then handle as a special case */
                        bad[0].stage = j;
                        bad[0].link = zeroins(j,b,n);
                        bad[1].stage = j;
                        bad[1].link = lower(j,zeroins(j,b,n),n);
/* bad[0] and bad[1] hold the fault labels for the non-stage n box, which */
/* must be checked to see if they block any s/d path; fault labels for the */

```



```

/* For all outputs in stage j (which must contain the link fault) */
    lbfault++;
    if ( i==n && j==n && b!=zeroin(i,a,n) && b!=lower(i,zeroin(i,a,n),n) )
        continue;
/* An LB-fault pair containing a stage n box fault and a stage n link fault. */
/* where the link fault is not in one of the two links attached to the */
/* faulty box, is not an LB-lossy pair, and need not be tested */
    oldloss = loss;
    if ( i==n ) {
/* If there is a stage n box fault then handle as a special case */
        bad[0].stage = j;
        bad[0].link = b;
        bad[2].link = zeroin(i,a,n);
        bad[3].link = lower(i,zeroin(i,a,n),n);
/* bad[2].link and bad[3].link are used here to hold the addresses of */
/* the two sources that must have only their primary paths checked */
        bads = 1;
/* there is only one fault label dealing with other than a stage n box fault */
        test(3);
    }
    else {
/* Case of no stage n box fault */
        bad[0].stage = i;
        bad[0].link = zeroin(i,a,n);
        bad[1].stage = i;
        bad[1].link = lower(i,zeroin(i,a,n),n);
        bad[2].stage = j;
        bad[2].link = b;
        bads = 3;
        if (verbose)
            printf("### (%2d,%2d) & (%2d,%2d)
& (%2d,%2d)\n", bad[0].stage, bad[0].link, bad[1].stage, bad[1].link, bad[2].stage, bad[2].link);
        test(2);
    }
    if (loss>oldloss)
        lblossy++;
}
}

for(a=0 ; a<N ; a++ ) {
/* Assume now that the link fault is in stage i: for each stage i output */
/* There can be no stage n box fault requiring special case treatment in */
/* this instance */
    for ( j=i-1 ; j>=0 ; j-- ) {
/* For all stages less than i */
        for ( b=0 ; b<N/2 ; b++ ) {
/* For the "upper" outputs of each box in stage j */

```



```

        lbfault++;
        oldloss = loss;
        bad[0].stage = i;
        bad[0].link = a;
        bad[1].stage = j;
        bad[1].link = zeroins(j,b,a);
        bad[2].stage = j;
        bad[2].link = lower(j,zeroins(j,b,a),a);
        bade = 3;
        if (verbose)
            printf("### (%2d,%2d) & (%2d,%2d)
& (%2d,%2d)\n",bad[0].stage,bad[0].link,bad[1].stage,bad[1].link,bad[2].stage,bad[2].link);
        test(2);
        if (loss>oldloss)
            lblossy++;
    }
}
}
printf("Number LB-lossy pairs = %d\n", lblossy);
printf("Number LB-fault pairs = %d\n", lbfault);

/* LL-fault pair section */
    loss = 0;
    for ( i=a ; i>0 ; i-- ) {
/* for all stages with links (i.e., excluding stage 0) */
        for(a=0 ; a<N ; a++) {
/* for all outputs of stage i */
            for ( j=1 ; j>0 ;j-- ) {
/* for all stages less than or equal to i that have links */
                for ( b=0 ; b<N ; b++ ) {
/* for all outputs of stage j */
                    if ( b<=a && i==j )
                        continue;
/* unless i=j, then for all stage j outputs greater than a */
                    llfault++;
/* count number of LL-faulty pairs */
                    oldloss = loss;
                    bad[0].stage = i;
                    bad[0].link = a;
                    bad[1].stage = j;
                    bad[1].link = b;
                    bade = 2;
                    if (verbose)
                        printf("### (%2d,%2d) &
(%2d,%2d)\n",i,a,j,b); */
                    test(2);
                    if (loss > oldloss)

```



```

        fault1 = fault2 = 0;
        for ( stage=n ; stage>=0 ; stage-- ) {
            link = (s2mask[stage])|(d2(-mask[stage]));
/* compute stage output of primary path */
            if ( faulty(stage,link) ) {
                fault1++;
                break;
            }
        }
        if ( labelcode==2 || (labelcode!=2 && s1=bad[2].link && s1=bad[3].link) ) {
/* Test secondary path only when appropriate, i.e., when the source is */
/* not connected to a faulty stage a box */
            for ( stage=n ; stage>0 ; stage-- ) {
                link = ((s2mask[stage])|(d2(-mask[stage]))) ^ 1;
/* compute stage output of secondary path */
                if ( faulty(stage,link) ) {
                    fault2++;
                    break;
                }
            }
        }
        else
            fault2++;
/* if the secondary path should not be tested, set fault2 as if the */
/* secondary were faulty so that if the primary is found to be faulty, */
/* the variable "loss" will be incremented, correctly indicating an */
/* s/d pair that cannot communicate */
        if ( verbose ) {
            printf("%2d to %2d ", s, d);
            printf(" primary: ");
            if(fault1>0) printf("faulty at stage %d.",stage);
            else printf("okay.");
            printf(" secondary: ");
            if(fault2>0) printf("faulty at stage %d.",stage);
            else printf("okay.");
            printf(" %s\n", (fault1>0 && fault2>0)?"LOSS":"NOLOSS");
        }
        if ( fault1>0 && fault2>0 )
            loss++;
    }
}

faulty(stage,link)
/*
** Check a stage output against the list of fault labels.
** If there is a match, return a 1; else return 0.
*/

```

```

*/
uint stage, link;
{
    int i;
    for ( i=0 ; i<bad : i++ )
        if ( stage==bad[i].stage && link==bad[i].link )
            return i;
    return 0;
}

lower(stage, link, n)
/*
** Generate the corresponding "lower" box output label
** given the stage number, label of the "upper" output,
** and the number of stages
*/
uint stage, link, n;
{
    if (stage == n)
        return(link ^ 1);
    else
        return(link ^ (1 << stage));
}

zeroin(stage, link, n)
/*
** Generate the "upper" box output label given the number of the
** stage containing the box, the "number" of the box within that
** stage (counting from zero with the "upper" box in the stage),
** and the number of stages
*/
uint stage, link, n;
{
    uint upper, lower;
    if (stage == n)
        stage = 0;
    upper = (link & (~0 << stage)) << 1;
    lower = link & ~(~0 << stage);
    return( upper | lower );
}

```

VITA

VITA

George Bunch Adams III was born in Wilmington, Delaware on March 1, 1956. He received the B.S.E.E. degree with distinction from the Department of Electrical Engineering, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, in 1978, and the M.S.E.E. and Ph.D. degrees in 1980 and 1984, respectively, both from the School of Electrical Engineering, Purdue University, West Lafayette, Indiana. As a graduate student he was a teaching assistant for the School of Electrical Engineering and a research assistant. His research interests include computer architecture, parallel processing, interconnection networks, and parallel processing algorithms.

Dr. Adams has co-authored eleven journal and conference papers, one book chapter, six technical reports, and has filed for a patent. He is a member of the Eta Kappa Nu, Tau Beta Pi, and Phi Kappa Phi honorary societies and the IEEE. From September 1981 to August 1983 he served first as Maintenance Officer and then as President of Purdue Pilots, Inc. Since August 1983 he has been a Research Engineer with the Research Institute for Advanced Computer Science (RIACS), located at NASA Ames Research Center, Moffett Field, California.

END

DTic

6-86