Wafer Level Systems: Theory and Applications

by

David C Blight

A thesis
Submitted to the Faculty of Graduate Studies
in Partial Fulfillment of the Requiements

for the Degree of

Doctor of Philosophy

Department of
Electrical and Computer Engineering
University of Manitoba

Winnipeg, Canada, 1994

ISBN 0-315-99095-3

Name David C Black

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

Electronics and Electrical Engineering
SUBJECT TERM

0544
SUBJECT CODE

U·M·I

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS
Architecture 0729
Art History 0377
Cinema 0900
Dance 0378
Fine Arts 0357
Information Science 0723
Journalism 0391
Library Science 0399
Mass Communications 0708
Music 0413
Speech Communication 0459
Theater 0465

EDUCATION
General 0515
Administration 0514
Adult and Continuing 0516
Agricultural 0517
Art 0273
Bilingual and Multicultural 0282
Business 0688
Community College 0275
Curriculum and Instruction 0727
Early Childhood 0518
Elementary 0524
Finance 0277
Guidance and Counseling 0519
Health 0680
Higher 0745
History of 0520
Home Economics 0278
Industrial 0521
Language and Literature 0279
Mathematics 0280
Music 0522
Philosophy of 0998
Physical 0523
Psychology 0525
Reading 0535
Religious 0527
Sciences 0714
Secondary 0533
Social Sciences 0534
Sociology of 0340
Special 0529
Teacher Training 0530
Technology 0710
Tests and Measurements 0288
Vocational 0747

LANGUAGE, LITERATURE AND LINGUISTICS
Language
- General 0679
- Ancient 0289
- Linguistics 0290
- Modern 0291

Literature
- General 0401
- Classical 0294
- Comparative 0295
- Medieval 0297
- Modern 0298
- African 0316
- American 0591
- Asian 0305
- Canadian (English) 0352
- Canadian (French) 0355
- English 0593
- Germanic 0311
- Latin American 0312
- Middle Eastern 0315
- Romance 0313
- Slavic and East European 0314

PHILOSOPHY, RELIGION AND THEOLOGY
Philosophy 0422
Religion
- General 0318
- Biblical Studies 0321
- Clergy 0319
- History of 0320
- Philosophy of 0322

Theology 0469

SOCIAL SCIENCES
American Studies 0323
Anthropology
- Archaeology 0324
- Cultural 0326
- Physical 0327

Business Administration
- General 0310
- Accounting 0272
- Banking 0770
- Management 0454
- Marketing 0338

Canadian Studies 0385
Economics
- General 0501
- Agricultural 0503
- Commerce-Business 0505
- Finance 0508
- History 0509
- Labor 0510
- Theory 0511

Folklore 0358
Geography 0366
Gerontology 0351
History
- General 0578
- Ancient 0579
- Medieval 0581
- Modern 0582
- Black 0328
- African 0331
- Asia, Australia and Oceania 0332
- Canadian 0334
- European 0335
- Latin American 0336
- Middle Eastern 0333
- United States 0337

History of Science 0585
Law 0398
Political Science
- General 0615
- International Law and Relations 0616
- Public Administration 0617

Recreation 0814
Social Work 0452
Sociology
- General 0626
- Criminology and Penology ... 0627
- Demography 0938
- Ethnic and Racial Studies 0631
- Individual and Family Studies 0628
- Industrial and Labor Relations 0629
- Public and Social Welfare 0630
- Social Structure and Development 0700
- Theory and Methods 0344

Transportation 0709
Urban and Regional Planning 0999
Women's Studies 0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES
Agriculture
- General 0473
- Agronomy 0285
- Animal Culture and Nutrition 0475
- Animal Pathology 0476
- Food Science and Technology 0359
- Forestry and Wildlife 0478
- Plant Culture 0479
- Plant Pathology 0480
- Plant Physiology 0817
- Range Management 0777
- Wood Technology 0746

Biology
- General 0306
- Anatomy 0287
- Biostatistics 0308
- Botany 0309
- Cell 0379
- Ecology 0329
- Entomology 0353
- Genetics 0369
- Limnology 0793
- Microbiology 0410
- Molecular 0307
- Neuroscience 0317
- Oceanography 0416
- Physiology 0433
- Radiation 0821
- Veterinary Science 0778
- Zoology 0472

Biophysics
- General 0786
- Medical 0760

EARTH SCIENCES
Biogeochemistry 0425
Geochemistry 0996
Geodesy 0370
Geology 0372
Geophysics 0373
Hydrology 0388
Mineralogy 0411
Paleobotany 0345
Paleoecology 0426
Paleontology 0418
Paleozoology 0985
Palynology 0427
Physical Geography 0368
Physical Oceanography 0415

HEALTH AND ENVIRONMENTAL SCIENCES
Environmental Sciences 0768
Health Sciences
- General 0566
- Audiology 0300
- Chemotherapy 0992
- Dentistry 0567
- Education 0350
- Hospital Management 0769
- Human Development 0758
- Immunology 0982
- Medicine and Surgery 0564
- Mental Health 0347
- Nursing 0569
- Nutrition 0570
- Obstetrics and Gynecology .. 0380
- Occupational Health and Therapy 0354
- Ophthalmology 0381
- Pathology 0571
- Pharmacology 0419
- Pharmacy 0572
- Physical Therapy 0382
- Public Health 0573
- Radiology 0574
- Recreation 0575
- Speech Pathology 0460
- Toxicology 0383

Home Economics 0386

PHYSICAL SCIENCES
Pure Sciences
Chemistry
- General 0485
- Agricultural 0749
- Analytical 0486
- Biochemistry 0487
- Inorganic 0488
- Nuclear 0738
- Organic 0490
- Pharmaceutical 0491
- Physical 0494
- Polymer 0495
- Radiation 0754

Mathematics 0405
Physics
- General 0605
- Acoustics 0986
- Astronomy and Astrophysics 0606
- Atmospheric Science 0608
- Atomic 0748
- Electronics and Electricity 0607
- Elementary Particles and High Energy 0798
- Fluid and Plasma 0759
- Molecular 0609
- Nuclear 0610
- Optics 0752
- Radiation 0756
- Solid State 0611

Statistics 0463

Applied Sciences
Applied Mechanics 0346
Computer Science 0984
Engineering
- General 0537
- Aerospace 0538
- Agricultural 0539
- Automotive 0540
- Biomedical 0541
- Chemical 0542
- Civil 0543
- Electronics and Electrical 0544
- Heat and Thermodynamics ... 0348
- Hydraulic 0545
- Industrial 0546
- Marine 0547
- Materials Science 0794
- Mechanical 0548
- Metallurgy 0743
- Mining 0551
- Nuclear 0552
- Packaging 0549
- Petroleum 0765
- Sanitary and Municipal 0554
- System Science 0790

Geotechnology 0428
Operations Research 0796
Plastics Technology 0795
Textile Technology 0994

PSYCHOLOGY
General 0621
Behavioral 0384
Clinical 0622
Developmental 0620
Experimental 0623
Industrial 0624
Personality 0625
Physiological 0989
Psychobiology 0349
Psychometrics 0632
Social 0451

WAFER LEVEL SYSTEMS:

THEORY AND APPLICATIONS

BY

DAVID C. BLIGHT

A Thesis submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

I hereby declare that I am the sole author of this thesis

I authorize the University of Manitoba to lend this thesis to other institutions or individuals for the purpose of scholarly research

I further authorize the University of Manitoba to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Abstract

Wafer Scale Integration (WSI) provides the potential to implement large parallel systems on a single wafer and offers advantages over other technologies in areas such as cost and performance. While WSI offers many advantages in the construction of large systems, its also poses many difficulties. We must accept the constraints of WSI and implement systems which are ideally suited for this technology. Our goal is to develop techniques which will allow us to implement a large parallel processing environment for general purpose computing utilizing WSI. This has been an active area of research for many years, and has met with many obstacles, most notable the utilization of a faulty environment for reliable computation. In order to exploit WSI for parallel computation, we must develop techniques which will incorporate fault tolerance. In this paper we examine two areas of difficulty, the reconfiguration and routing in a faulty network. We attempt to exploit the inherent disorder in the fault environment, and develop techniques to offer reliable and efficient communication with it.

Traditionally fault tolerance in WSI processor arrays has been achieved by reconfiguring the faulty processor array, and/or by utilizing adaptive routing techniques to route messages around faulty elements in the network. Reconfiguration normally attempts to alter the connectivity of a network so that the resulting network is a regular topology (for example a k-ary n-cube). In order to perform such reconfiguration, either spare processing elements are utilized, or a subset of the original network is utilized to create the desired topology. The main limitation of these reconfiguration techniques is that not all functional processing elements in the network are utilized, and restrictions on allowing faults to only occur in processing elements limits the effectiveness of these techniques. Although adaptive routing algorithms can be used to allow communication in a defective environment, without reconfiguration the bandwidth of these networks is not sufficient to provide reasonable performance as congestion caused by faults will dominate network latency.

In this thesis we discuss an alternative to the traditional approach of reconfiguration and adaptive routing. We first reconfigure our faulty processor array, not with the goal of producing a regular topology, but rather to increase the bandwidth of the network (increase the conductivity of a network defined by percolation theory). This reconfiguration will provide us with a network which is disordered and irregular but has sufficient connectivity to allow the implementation of message passing techniques (such as store and forward or wormhole techniques). In the second part of our techniques we implement modified adaptive routing algorithms which can provide deadlock free, livelock free and starvation free routing between most elements in the reconfigured array. This allows regular and irregular topologies to be implemented on the processor array, and provide reliable and effective communication between required processing elements.

Acknowledgements

This thesis, as with all research, is not the sole work of a single author, but results from collaboration and interaction with many people. I would like to take this opportunity to thank all the people who have contributed towards this thesis.

I would first like to thank my advisor, Dr. Robert D. McLeod, for all his help and guidance over the years. I owe Bob thanks, not just for his contributions and help with this thesis, but for his help and guidance with almost my entire university education. It is Bob's contributions and encouragement which have most strongly shaped both this thesis and my education. If anyone truly deserves credit for this thesis, it would be Bob.

I would also like to thank the member of my thesis committee, Dr. Howard C. Card, Dr. Witold Pedrycz, Dr. Attahiru Alfa, and Dr. Ted Szymanski, for taking the time to be on my thesis committee, to read both my candidacy exam and thesis, and for all their inputs during this thesis.

I would also like to acknowledge the help and encouragement received from fellow graduate students over the years. Without their work and friendship, this research would not have been so pleasurable. I would like to thank Zaifu Zhang for his discussion on matters related to both research and life in general. Dean McNeill desires special thanks for helping maintain the computer network and giving me more time to work on research. Almost all graduate students have contributed indirectly to this research, and while I can not thank them all, I would like at least to mention Ken Ferens, Tapas Shome, and Budi Rahardjo.

I would also like to thank friends and family, who have supplied encouragement and support over my many years of university education. Without the encouragement of my parents, brother Stephen, Aunt Betts, and friend Bill Emslie, I would have never finished this thesis. I would also like to especially acknowledge the support of Marzena Slabon who has always been willing to help.

And finally I would like to acknowledge the support of the Canadian Microelectronics Corporation for their support through the equipment loan programs. I would also like to acknowledge the financial support of the Federal Government's Networks of Centres of Excellence Program Micronet.

Dedication

I would like to dedicate this thesis to the memory of my Grandfather
W. C. Blight

Table Of Contents

List of Figures

List of Tables

CHAPTER 1: Introduction

Wafer Scale Integration has long been an active area of research. The goal has been to utilize the entire surface of a wafer for the implementation of a single system. The advantages of this technology include potential cost and performance advantages over other technologies. Despite over thirty years of active research into problems associated with utilization of this technology, very few commercial systems have been made.

There are a myriad of problems associated with WSI. One of the most difficult problems with utilizing this environment, is developing systems which are able to operate with the high defect rates typically associated with this technology. Unlike traditional Integrated Circuits (ICs) which are small enough that we can fabricate fault free circuits, the enormous quantity of devices which can be fabricated on a single wafer, make it impossible for even the best fabrications lines to create fault free designs. Any design targeted for WSI must include fault tolerant techniques in its design.

Our goal in this thesis is to investigate some of the problems associated with implementing large systems in the highly defective environment of WSI. The problem we will look at will be with implementing a large general purpose parallel processing system. We will investigate the properties and requirements of this type of system, and develop a strategy to implement it using WSI technology. The focus of this research will be on the problems associated with communications in large networks in the presence of faults. We will deviate from the traditional approach of attempting to reconfigure the network into a regular topology, and will instead focus our efforts on implementing routing in a disordered environment created by defects.

In Chapter 2 we will present an introduction to WSI, and discuss both the advantages and disadvantages of this technology. We will review some of the past research in this field, and discuss some of the commercial applications which have been attempted. We will also discuss some of the basic concepts of parallel processing, and the trade-offs related to implementation technology.

In Chapter 3 we will introduce our model of the parallel processing system we will be studying in this thesis.

In Chapter 4 we will analyze large processors arrays using techniques borrowed from the study of large networks, specifically percolation theory. We will use the result to implement a non-deterministic adaptive routing algorithm, which will route messages in the highly disordered WSI environment.

In Chapters 5 and 6, we will attempt to improve communications in the networks, by utilizing a routing technique known as wormhole routing. This technique generally offers better performance over the traditional routing approaches, however is highly susceptible to deadlock. In these chapter we will try to exploit the disordered nature of the processor arrays, and apply reconfiguration within a disordered network. We will implement a routing algorithm which can successfully route in this environment.

CHAPTER 2: Wafer Scale Integration

Wafer Scale Integration (WSI) has long been proposed as an alternative to the traditional VLSI technology for implementing electronic circuits. Integrated Circuits (ICs) are electrical circuits fabricated such that all components are on a common die of silicon. Typically ICs are fabricated by diffusing and depositing materials onto a silicon wafer in such a way as to produce the desired circuit components (typically transistors) and connections. A typical wafer used to fabricate complex ICs will be approximately 15 to 20 cm in diameter, and will typically contain hundreds of identical copies of an individual IC, each of which is usually approximately 1 cm^2 for a large VLSI circuit[22]. After fabrication, each wafer will be sliced into individual dies, each of which are placed into chip packages. Defects in the ICs will render a portion of the ICs unusable, and hence each IC must be tested, and those with operational characteristics not within the specifications of the original design are discarded.

The traditional IC approach has improved both the density and performance of designs over the last 30 years by increasing the number of transistors in an IC and by improving the switching speed of devices and interconnects. Most of these improvements have come from decreasing the feature size of devices, and by using larger die sizes. We are currently approaching the fundamental limits for transistor sizes, in which the non ideal, parasitic, and quantum effects begin to dominate the behaviour. As a result, it is generally accepted that not much improvement which be made in further reducing device sizes, and most improvements will come from increasing the physical area of an IC.

The goal of Wafer Scale Integration is to utilize the entire wafer surface for a single circuit or system. Although this is not a new idea, many difficulties exist in attempting to fabricate and utilize entire wafers. Numerous gains are expected from technology which make its pursuit worthwhile[23].

1. Reduced manufacturing costs by reducing the number of distinct components in a large system.

2. Improved performance by reducing the need for off-chip communication. Reduced noise on interconnects both improves performance and makes more reliable circuits.

3. Lower power consumption by reducing the need for high power off-chip I/O drivers.

4. Improve reliability by reducing the need for inter-chip connectivity.

In this chapter, we will discuss the advantages, disadvantages, and applications of WSI technology which have been researched and exploited to date. The focus of this thesis will be on the exploitation of WSI for parallel processing applications, and we will be primarily interested in large processor arrays. Our goal is to produce a general purpose parallel processing computing system on a wafer, by replicating an array of processors on the wafer surface.

2.1 Difficulties with WSI

WSI has been an active area of research for over thirty years, however very few commercial products have utilized this technology. In this section we will review some of the major difficulties which have prevented the widespread use of this technology.

2.1.1 Defects and Yield

A defect in a silicon wafer may be described as a significant deviation in the physical state of the wafer from the desired state or alternatively as a physical imperfection in the wafer sufficiently large to change the characteristic of the wafer. Typically defects result from the less than ideal conditions available during the fabrication of wafers and the circuits on them. Impurities may be introduced into the wafer, misalignment of different processing steps may produce incorrect geometries, and damage may occur during the handling of a wafer.

Defect avoidance refers to techniques applied during the fabrication process in order to reduce the occurrence of defects. Examples of this include the reduction of impurities in chemicals used during fabrication and close monitoring of wafers during fabrication. Defect tolerance refers to techniques used to make wafers less susceptible to the effects of defects. A typical example of this is the use of design rules which specify constraints on design geometries.

A fault is an incorrect behaviour of a signal within a circuit. A fault may be the manifestation of the defect in the operation of circuit, or may be caused by external influences such as radiation. Many fault models have been developed to abstractly model the most common faults[26][80]. For example:

- Stuck-at faults assume that signals (wires) are forced to constant values.
- Stuck-open and stuck-closed faults assume that devices (transistors) are forced to permanently off or on states.
- Bridging faults assume that signals (wires) may be shorted, forcing the same values on two lines.
- AC-faults assume characteristics such as rise and fall times of a component are affected.

Unlike defects which are assumed to be permanent, faults may exist for different time intervals. Generally faults are assume to be described by one the following three categories:

- Permanent faults: once they occur they remain in the circuit permanently or until the circuit is repaired.
- Intermittent faults re-occur in the circuit for finite periods of time. Intermittent faults remain active long enough that testing circuitry can determine that a fault exists.
- Transient faults remain in the circuit for a finite but brief period of time. Transient faults typically only exist for one signal transition.

Although defect avoidance and defect tolerance techniques are applied during wafer fabrication and design, we must assume that defects will always be present and that if we wish to utilize a wafer, fault tolerance techniques must be applied to the system being implemented.

2.1.2 Architectural Issues

One of the major difficulties of utilizing WSI technology, especially as device sizes decrease, and integration levels increase, is to decide the best method to utilize the wafer surface for different types of systems architectures. Many conventional architectures are designed with different constraints in mind than those imposed by WSI. Although much larger systems than those possible with WSI have been developed utilizing other technologies, much work has to be done in developing systems architectures which will suit the two dimensional local connection constraints of WSI. As the level of integration increases, we have to examine architectures which will scale with network size. These designs have to be suitable for implementation even when the number of devices increases.

A number of different topologies for parallel systems have been proposed and many have been implemented. These topologies are often developed for systems which have different constraints than those of WSI. The following is a brief list of some of the more popular topologies:

1. Mesh: This is the simple two dimensional array, with four nearest neighbour connections. This is one of the most popular topologies used in commercial systems.

2. Hex array: This is an extension to the mesh but allows connections to six nearest neighbours. Similar networks may be constructed with eight nearest neighbours.

3. k-ary n-cubes: This is an abstraction of meshes to higher dimensions than two. n represents the number of dimensions in the topology, and k represents the number of nodes in each dimension. Such a network will contain k^n nodes, and nk^n edges. A 10 by 10 mesh is also considered a 10-ary 2-cube.

4. Hypercube: This is an extension of the unit cube to n-dimensions, and the binary case of the k-ary n-cubes. This topology offers a high degree of connectivity and a small diameter n. Unfortunately, it requires a large number of connections per node. The degree of the network also increases with size, which makes it impractical for most applications. Hypercubes also do not have a planar representation and hence require long wires for large implementations. The lack of scalability prevents this topology from being used except for small networks.

5. Cube-connected cycles: This is an extension to the hypercube where each vertex is replaced by a cycles of n nodes, which transforms the network to a constant degree (3) network. Like hypercubes, these networks are not planar.

6. Star connected networks: This network has all nodes connected to a common node. The connectivity on the central node makes this topology unsuitable for large networks,

7. Bus based networks: Bus based networks are very commonly used in Local Area Networks (LANs). Although they are the easiest to implement, they do not scale very well. Common implementations include ethernet and token ring.

In this thesis, the mesh will be examined almost exclusively since it is the only topology with a planar representation.

2.1.3Power Dissipation

One of the earliest problems recognized with WSI is the problem of power dissipation. With very high densities of devices on a single wafer, a large amount of heat is produced. This heat has two potentially damaging effects[14][59]. The major difficulty is when stress is placed on the packaging of the wafer. Although not much stress will exist internally in the wafer due to the uniform nature of the wafer, stress will be placed on the package which holds the wafer. A second problem is that if densities of devices increase dramatically, the heat produced from computation may be sufficient to melt the silicon substrate.

Two approaches have limited the harmful effects of heat dissipation in WSI. Firstly, good packaging technologies have been developed which can remove heat from the wafer, and secondly heat dissipation can be reduced in circuits, by utilizing lower power circuits (such as the switch to CMOS from NMOS and bipolar technologies). Asynchronous designs also offer the potential to lower heat dissipation in circuits[41][78].

2.1.4Fabrication Techniques

When we refer to fabrication difficulties we are normally referring to two issues. One is creating a fabrication process capable of implementing wafers which have sufficiently low defect rates, and the other is doing so with reasonable cost. Most of the problems of fabricating WSI devices are the same as those that are pertinent to IC fabrication. In addition however, there are additional processing issues which must be addressed for WSI.

A common technique used in reconfiguration of WSI devices, is to use laser programmed fuse and antifuse techniques to reconfigure wafers[2][15]. These techniques allow the possibility to add connections, and also to remove connections after the main part of the fabrication process. Although these techniques have been used for a considerable time, and much progress has been made, the additional non standard fabrication techniques make these processes more expensive. In addition the fuses and antifuses all introduce a performance cost, as the devices are slower than traditional connections.

One of the current trends in IC manufacturing is to fabricate larger wafers. Recently many companies have been moving towards utilizing 8 inch wafers. This offers the potential to imple-

ment both larger ICs and to increase the number of devices per wafer. This increase in wafer size reflects the ability to better manufacture larger wafers.

It is possible that alternative technologies to silicon will also play a role in future. Technologies such as BiCMOS offer the potential to drive larger and longer lines which will be useful in the design of systems requiring longer interconnects or higher fanout. Gallium-Arsenide, may offer the ability to integrate optical structures such as optical interconnects.

2.1.5Testing

As with almost all other parts of the design and implementation of WSI systems, most of the problems associated with conventional VLSI design are almost always present in the design of wafer systems. The problems are normally compounded however in WSI, because of the increase in size of the system. This is particularly true for problems such as testing, which are affected by the size of problems (these problems are usually NP complete). Testing is concerned with determining if a produced systems meets both the functional and performance parameters intended. The problems and costs associated with testing a digital system increase exponentially with system size.

Testing is a critical component of the design of any electronic system[25][80][81]. Many techniques have been developed to help improve the testability of designs. One of the most significant developments was the concept of Design for Test (DFT), and use of Built In Self Test (BIST). DFT implies that testing issues are taken into consideration while the system is being designed in order to increase the testability of a design. Testability is usually measured as the ease in which a fault may be found in a circuit, and is related to the controllability and observability of internal signals in a circuit. BIST utilizes test circuitry which is added to a design in order to test the design. BIST is normally applied by dividing the circuit into combinational logic sections, and utilizing modified latches in the design as test circuitry. By having the BIST circuitry generate pseudo random patterns which are applied to the combinational logic blocks, and a signature analyser to compress the output patterns, a signature is generated which can be used to compare actual outputs to expected outputs. Although aliasing can occur in which faults go undetected, these techniques will typically detect nearly all faults.

The Boundary Scan (BS) methodology, standardized in the JTAG[36] standard is a testing methodology which is gaining commercial acceptable and is applicable to processor arrays in WSI. The idea of BS is to add latches around the external connections of components in a design, and allow these latches to be serially connected together. This allows a convenient and standardized way to apply test vectors to components in a design, and is compatible with BIST techniques.

Testing of asynchronous systems is not as straightforward as testing of synchronous systems which is where most of the formal theory and methodologies have been developed [49]. With asynchronous systems it is much more difficult to test a design as it has more complex data movements, and parameter testing will be much more a concern.

2.1.6Power Distribution

The large amount of devices on a single wafer create problems of power distribution[37][39]. As was mentioned previously, it is important that the entire wafer not be totally synchronized, as this would create large transient currents during the operation of circuits, especially in CMOS circuits where the most power is dissipated during signal transitions. Synchronous systems will have large power spikes at clock transitions. Even in the case of asynchronous designs, it is still important to ensure an even distribution of power, and to insure that the power lines are noise free. It is important that the power distribution be viewed as a distributed network, and not the lumped element model usually employed for VLSI design.

An additional problem is also encountered in power distribution on wafers when reconfiguration is performed to isolate faulty elements. It may be necessary to isolate a faulty element not only logically from the network, but also from the power and ground supplies. If a faulty element contains a short from power to ground, the behaviour of neighbouring devices may also be affected. It may be necessary to include switches not only in the signals lines, but also the power and ground lines.

2.2 WSI Architectural Constraints

The large area of the silicon wafer and the high packing density of VLSI circuits offers the potential to implement large designs on a single wafer and at the same time imposes constraints on the designer. In this section we will discuss the architectural issues related to WSI implementation which will impose limits on the systems utilizing WSI technology.

As discussed in the previous section, it is highly improbable that a wafer will be fabricated with no defects. It is imperative that if a design is to utilize the entire wafer surface, that fault tolerant design methodologies be employed. Two of the most common fault tolerance methodologies proposed for WSI system include replication of hardware and information to provide redundancy which may be used to mask errors[69][70]. An alternative approach is to reconfigure the faulty circuit to bypass faulty components.

A second constraint on WSI systems is the length of wires. While on the relatively small area of a typical IC it is possible to have wires running the entire length of the die, on a wafer, RC effects prohibit the use of arbitrary long lines. This is an important consideration, as it will affect our choice of network topology, and prevent the use of long lines for global signals such as clocks.

Another constraint that we must observe is that we wish to implement scalable systems. We wish to develop techniques of implementing and utilizing hundred, thousands, or more processors in a single system. In order to accomplish this we must develop techniques and algorithms which will allow us to exploit this large degree of parallelism without being limited by the number of processors. These systems must work for both implementations with a small number and a large number of processors.

Based on these observations of WSI systems, any large design implemented using this technology will need to obey the following constraints:

1. The topology of the fabricated circuit must be planar. This is a restriction placed on the design by the two dimensional nature of the wafer.

2. The topology of the system architecture must have a two dimensional (planar) representation. Although higher dimensional structures are useful for many applications, we must fabricate a circuit which does not contain any long wires, or long interconnect networks. If we attempt to put higher dimensional structures on a planar surface, we will be forced to devote a portion of the physical area to the interconnect network which will be proportional to the network size. This arrangement will not scale.

3. The basic structure of the circuit should be regular.

4. All network operations of the system should be based on strictly local information and algorithms.

2.3 WSI applications

The first attempt to exploit WSI came in the early days of IC technology when Texas Instruments (TI) attempted to fabricate small circuits on a wafer[14]. Defect rates and low yield limited IC technology in those days to circuits with approximately 10 gates. Discretionary wiring was used to connect clusters of working gates together in an attempt to create larger circuits (this is an early form of a gate array). Water cooling was used for heat dissipation. A 32-kilobit memory was implemented using this technology. An expensive fabrication process, and lack of demand for the level of integration at this time prevented this technology from catching on and becoming a commercial success. Alternative technologies where more suitable for the industrial needs at that time, and were more economically viable.

One of the earliest commercial attempts to utilize WSI was Trilogy's attempt to create an IBM-compatible system 360/370 mainframe[59][14]. After considerable work in developing WSI products, Trilogy was able to develop a package for WSI which was able to dissipate 50 W/cm^2. The biggest difficulty was the defect rate of the wafers. The required redundancy they had to incorporate into a wafer, lowered the potential cost and performance advantage of WSI. Although Trilogy eventually abandoned wafer scale products, it was not due to the failure of wafer scale technology, but primarily to the cost advantage of other technologies.

A task flow architecture has been proposed and studied[35] with WSI implementation in mind. The goal in this project was to develop a large computer constructed from a a large number of cells connected by a network for intercell communication. In their proposed WSI implementation, a large linear array of functional cells is constructed from the wafer, and used for computation. No actual machine or prototype was constructed.

Another common application for WSI has been systolic arrays[51][52]. This type of architecture is fairly well suited for WSI environments, as the interprocessor communication is fairly

simple. The difficulty in constructing these types of networks has primarily been with reconfiguration so that processors have a link to an appropriate neighbour after reconfiguration. The difficulty in the reconfiguration for these networks is that we usually need to reconfigure a 2 dimensional mesh from the faulty mesh. A large number of redundant spares, or complex reconfiguration schemes are needed to ensure the fault tolerance required to handle typical defect rates.

The most successful utilization of WSI technology has been associated with memory devices. The Spiral Array was proposed in 1978 to create a spiral of shift registers. In 1984, the Wafer Scale RAM was announced in which a 256K X 6 RAM was constructed using 3um CMOS. Other attempts include the Full Wafer MOS RAM, and Memory by Configuration Logic. Currently some commercial memory products are available. Inova Microelectronics has available 1 Mbit and 8 Mbit static RAMS, and Fujitsu has a 200-Mbit memory available.

WSI technology is not limited to strictly electronic systems. Work has been done in utilizing WSI for Transducer arrays in which micromachining technology is applied to WSI to produce three dimensional machines[15]. Many of the difficulties in building these system in WSI are similar to those of digital systems. Redundancy and reconfiguration are still required to avoid defects.

Although the history of WSI has suffered many failures, much has been learned about utilizing WSI technology, and advances in many areas are making WSI a more attractive technology for the future. It is unlikely that WSI will be the sole implementation technology for the future, but it will almost certainly have a place for specific types of applications. Although most of the WSI projects have not been commercial successes, they have not failed for technological reasons. In most cases, the products simply did not offer any significant cost or performance improvements over more conventional technologies.

Today there are some new alternative technologies related to WSI. Multi Chip Modules (MCM) are a relatively new technology where ICs are mounted onto a interconnect substrate.

2.4 Multiprocessor Architectures

WSI offers the potential to improve performance of large systems, but we also need to investigate new architectures and techniques in the design of systems which will be able to effectively utilize the available WSI environment. The large amount of resources available on a single wafer would not be best utilized by a traditional single threaded computer architecture, and only a small portion of the available circuitry would be utilized at any time. Parallel processing attempts to improve computation speed by utilizing concurrent hardware to reduce execution time, rather than increasing processor speed which has been the primary source of increased computer performance in the past. In this section we will briefly discuss some of the different approaches which are commonly used. More complete coverage of parallel processing architectures can be found in [53].

2.4.1 Shared Memory

The shared memory model for parallel computation assumes that we have a distinct collec-

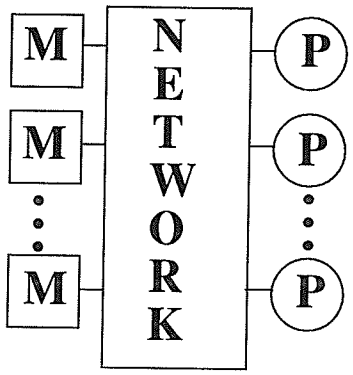


Figure 1 Shared Memory Network

tion of processors and memory, and a interconnection network which allows all processors to access all memory. This is shown in Figure 1. On the right side of the network we have a set of processors, on the left side we have a set of memory components. This type of network allows each processor to access any of the memory elements. This model is one of the most basic and is often used as a programming model for parallel processing as it is one of the most conceptually simple models, and closest to the sequential processing model which is the model most programmers are familiar with.

One of the difficulties with this model is that it is not easy to implement, especially for large networks. The network is usually constructed as a Multistage Interconnect Network (MIN) which is constructed by using multiple stages of interconnect elements to create a network which can connect any input (processor) to any output (memory). The size and complexity of the network will limit the number of simultaneous connections (access) which can be made between processors and memory.

This approach is useful primarily for a model of parallel processing, and for small networks. Unfortunately the complexity of the network will increase rapidly, and the performance will degrade as the network size is increased.

An example MIN and a routing algorithm for it are discussed in Appendix B.

2.4.2 Direct networks

An alternative approach to parallel processing is the direct network. In this model each processor has a local memory, and the interconnect network is used to provide communication between processors. If a processor wishes to access some data stored in a memory of another processor it must send a message to that processor requesting the information. The processor with the data must accept all requests and respond with a message containing the desired information. Each processor in the network must be able to receive and forward messages which are not destined for

them. An example of this type of network is shown in Figure 2.

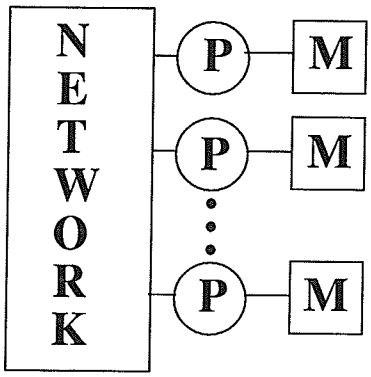


Figure 2 Direct Network

2.5 WSI Processor Arrays

WSI offers an environment suitable for the implementation of many types of systems. In this thesis we will only be concerned with the implementation of parallel processor systems, which will be implemented using a direct network for interprocessor communication. Our system will be constructed by fabricating a regular array of processors on a wafer, and implementing a nearest neighbour connection scheme. These networks are referred to as processor arrays. An example application suitable for WSI processor arrays is discussed in Appendix A.

It is the structure of the interconnect network which will characterize its performance. It is also the nature of this network which will separate the direct and indirect networks. Whereas in indirect networks a network is constructed in which the goal is to communicate between input and output nodes, and all other nodes are used to provide connectivity between these nodes, direct networks make more effective use of resources by utilizing each node for both processing and communication. In an indirect network, the focus of the interconnect network is to provide communication between processor and memory using the shortest available paths. In direct networks, each node is an input and output node, and hence must be able to communicate with each other. In this section we will briefly examine some topologies for direct networks.

2.5.1 Linear Arrays

The most simple processor array to construct is the linear array, which is essentially a one dimensional string of processors. Although this network has the simplest interconnect network for direct networks, and an extremely simple routing algorithm, these networks have a high diameter (the maximum distance between processors), and a low bisection width (minimum number of connections to remove in order to divide network). This type of network is useful for constructing devices which utilize one dimensional networks such a memory devices and simple functions (auto

- correlation). It is also important to realize that we can create a one dimensional network from a

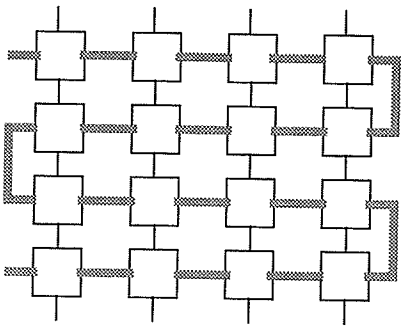


Figure 3 One Dimensional Network

higher dimensional physical layout. In Figure 3 a linear array is constructed from a two dimensional layout.

2.5.2Two Dimensional Structures

By far the most common topology for processor arrays is the two dimensional network in which processors are laid out along a two dimensional Cartessian coordinate system. The simplest of these arrangements is referred to as a mesh, and is shown in Figure 4.

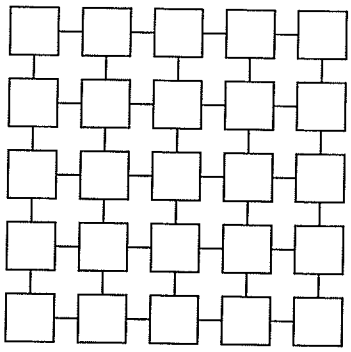


Figure 4 Mesh

The mesh has the following characteristics:

1. It is regular. A design for a processor can easily be replicated in a regular pattern of the wafer.

2. Simple routing algorithm. It is trivial to find the shortest paths between two processors in this network. This is important in implementing local routing functions for these networks.

3. Diameter $2\sqrt{N}$ where N is the number of processors in the network.

4. Bisection Width $\sqrt{N}$.

5. The mesh is clearly planar.

The mesh is not the only two dimensional structure which can be implemented. Some alternatives are:

1. Butterfly and Shuffle Exchange. These networks are useful since they have a lower diameter, but require long wires to implement.

2. Hex connected Networks. Instead of utilizing four nearest neighbour connections, six may also be used. This creates a hex connected network which has similar properties to a mesh, however has higher vertex degree.

3. Torus. A torus is an extension to the mesh in which the edge connections are wrapped around the network. This has the advantage that each node will have the same degree, but will require long wires to implement. In addition we will utilize the edge connections in a mesh for off network communication.

2.5.3 Higher Dimensional Topologies

There are of course other topologies with higher dimension than two. The 3 dimensional mesh, and hypercubes are examples. The k-ary n-cube is a generalization of regular high dimensional topologies. k refers to radix of the network (the number of nodes along each dimension) and n is the dimension of the network (vertex degree). These networks will have higher degree, higher bisection width, and lower diameter, however they are not as suitable for WSI implementation as the mesh architecture.

2.6 Reconfiguration Techniques

Reconfiguration is the process by which the interconnects in a network are restructured. In the case of WSI processor arrays, the goal is to change the connectivity of a network so that the presence of faults can be negated. In this section we will briefly review some of the common reconfiguration techniques which have been proposed. We will only look at reconfiguration techniques for two dimensional meshes as these are the networks we will be investigating in this thesis. Many of these techniques are applicable to other topologies; in addition other techniques exists for other topologies.

All the reconfiguration schemes mentioned in this section attempt to create a regular topology out of the faulty array. The targeted structure may be a one or two dimensional structure.

2.6.1 One dimensional Techniques

Some of the simplest reconfiguration techniques, and the most successful are those in which a two dimensional mesh with faulty processors is reconfigured into a linear (one dimensional) array. This technique is particularly useful when the arrays are used in memory type structures, and for simple systolic and pipelined processor networks.

There are three basic types of linear reconfiguration which have been proposed[62]. These techniques are classified according to the type of algorithm used to find the linear sequence of functional processors.

The patching method is a divide and conquer technique in which the wafer is divided into patches, which are non overlapping rectangular groups of processors. Each patch is then reconfigured into a linear array. Once all the patches have been reconfigured, the linear arrays in each patch are then connected together to form a single linear array which spans the network. Hierarchical versions of this type of algorithm have also been proposed. These algorithms can obtain total reconfiguration of all functional processors, but the produced linear array will not be optimally reconfigured, as the local reconfiguration in each patch does not optimize according to distribution of functional processors in neighbouring patches.

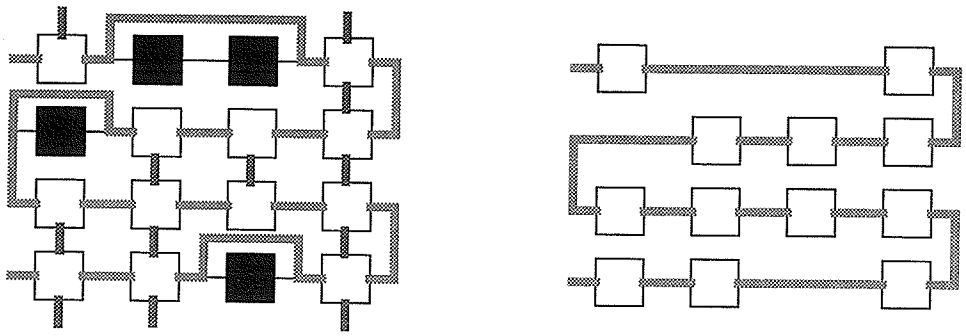
The tree approaches to reconfiguration start by finding a spanning tree which covers the functional processors in a network. From this tree, the branches are linearized to produce linear arrays. In this approach not all functional processors will be utilized in the final linear array. These techniques may produce a slightly more optimized reconfiguration than the patch method.

A third category of reconfiguration schemes is referred to as spiral approaches. In these techniques an initial functional processor is selected, and a neighbouring processor is selected to form the next element in the linear array. If a processor is reached which has no more non selected functional processors as neighbours, back tracking can be performed, and the algorithm continues along another path. Simple rules have been proposed in which the next processor is selected, and these may create structures which resemble spirals. Techniques have been proposed which can have both ends of the linear array on border processors of the network. Like the tree based approaches, these techniques will not in general utilize all functional processors in the reconfigured network.The advantage of these techniques is that the reconfigured array will not be required to have any long bypass connections as required by the patch method and tree methods.

2.6.2Diogenes

Diogenes reconfiguration[71] is another reconfiguration technique for producing linear arrays. Reconfiguration is accomplished by utilizing bypass interconnects and switches which may be set to bypass and isolate faulty elements. Figure 5 shows the reconfigured (horizontally) one dimensional array, and the corresponding logical linear array.

The Diogenes reconfiguration scheme has the advantage that a 100% harvest rate of function processors can be achieved, and that only a simple reconfiguration scheme is required. The disadvantage of this scheme is that in synchronous systems, there is an unbounded delay between adjacent processors in the reconfigured network. We will utilize an extension of the Diogenes reconfiguration scheme is later chapters when we look at reconfiguration of two dimensional arrays.



(a) (b)

Figure 5 Diogenes Reconfiguration

2.6.3Interstitial Redundancy

Reconfiguration is more difficult when the desired reconfigured topology has a higher dimensional topology than the linear array. Interstitial redundancy attempts to alleviate the problem by introducing spares processors into the network. The operation of faulty processors is transferred to these spare processors, and the spares are reconfigured to be able to communicate with the appropriate neighbours.

Interstitial redundancy can be used to achieve fault tolerance for up to approximately 10% faults (90% yield rate)[62]. Although this technique allows successful reconfiguration, long wires, and complex reconfiguration circuitry are required. A less than 100% harvest rate of function processors is achieved.

Alternative reconfiguration schemes exist which also use techniques of redundant spares. Extra columns and rows of spares can be added to a network, and these spares can be reconfigured to replace faulty elements. These techniques all suffer the same problems as interstitial redundancy. An alternative approach is to eliminate row and columns from the network which contain faulty elements. These techniques also achieve less than perfect harvest, and do not scale well to large networks.

2.6.4Connection Redundancy

It is also possible to perform configuration on the communication connections in a network as opposed to the processors. We will examine this in chapter 6.

2.7 Routing Techniques

In direct networks, communication may be required between processors not connected by a physical channel. Information can be sent in packets through the network to their destination. Routing techniques are classified into two basic categories: Circuit Switched (CS) and Packet Switched (PS).

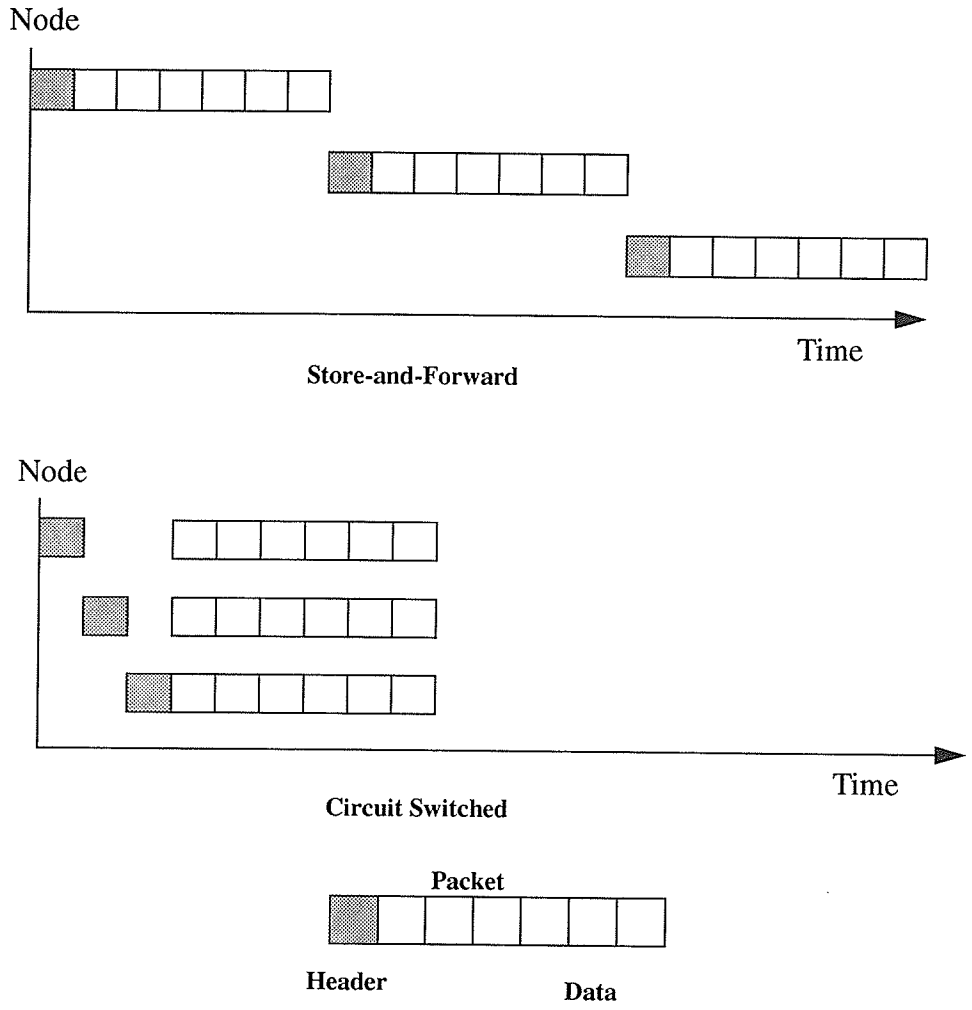


Figure 6 Comparison of Store and Forward and Circuit Switched Routing

Circuit switching techniques allow communication in networks by opening a channel between the source and destination nodes. A channel is a set of interprocessor connections which together form complete path. A channel is opened by reserving (allocating) all the connections required to form a channel. Once a channel is opened, a direct connection exists between source and destination nodes, and information is transferred without any buffering between the two nodes. Once the message has been sent, the channel may be closed, and the reserved connections which formed the channel may be freed for other channels to use. More sophisticated circuit switched networks allow multiple channels to be multiplexed on single connections. This technique is quite common in telecommunication applications where high bandwidth connections are used, and only small bandwidth signals need to be transmitted. Circuit switched techniques are used in applications where a guaranteed bandwidth must be maintained. Typically there are non constant delays in establishing a connection, but once the channel is opened, a nearly constant delay is available

for transferring information.

An alternative communication technique for networks is to use packet switched networks. In these techniques, a message is divided into packets which are transferred between nodes in the network. Packet switched networks offer a different type of performance characteristic than CS ones. Whereas CS networks experience network congestion and blocking in establishing a connection, PS networks do not require any time to establish a channel, but instead experience a delay on each packet of a message which may not be constant. Depending upon the nature of the PS routing algorithm, packets may be buffered at intermediate nodes. An example of a packet switched network is the common TCP/IP email system, in which electronic mail messages are transferred between computers as the message travels towards its destinations.

The packet switched networks were traditionally used in computer communication networks, in which the network latency was not a critical issue. Today however, it is becoming more common to use these techniques even in the telecommunication industry as the routing techniques become more advanced, and the communication links faster. ATM is an example of a telecommunication protocol using packets switched techniques.

Packet switched networks have evolved over the years, and are now applied to many different types of systems. As a result there have been many different types of PS algorithms developed for the different types of networks.

2.7.1 Store and Forward

Store and forward routing is traditionally one of the most common routing techniques employed in large PS networks. A message (which can be of arbitrary size) is divided into fixed sized units called packets. Each packet contains both routing information and data. A message is transferred between two processors by transmitting the packet through the connection, The packet is buffered at the receiving processor, and once the entire packet is received, it may be transmitted to another processor on its way to its intended destination. Most networks which implement this type of routing will utilize a fairly large buffer to hold a multitude of messages at one time.

The network latency for Store and Forward routing can be approximated (ignoring congestion and blocking) by (L/B)D where L is the size of the packet, B is the channel bandwidth, and D is the length of the path through which the packet will be routed. This formula is meant only to comparatively illustrate the latency of routing algorithms and is not a precise description of network latency. The effects of queuing delays which will dominate performance characteristics will be analysed in subsequent chapters. A comparison of circuit switched and store and forward routing can be seen in Figure 6. In packet switched networks, the time required to deliver a message (with no other traffic in the network) will be the product of the time to transfer a packet times the length of the path. For large networks, this may create an unacceptable delay. Circuit switched routing offers better latency in networks with no other traffic, although they will not be able to offer this performance in large networks with other traffic. Two processors attempting to send messages across a network using circuit switched techniques will likely experience a contention for resources and the probability of being blocked in attempting to allocate a path through a network will be re-

lated to network size and congestion.

2.7.2 Virtual Cut-Through

The virtual cut-through routing technique[23] was proposed as a routing methodology which would attempt to combine the best characteristics of packet switched and circuit switching routing techniques. In this technique, a packet is only stored at a node if the output channel through which it would leave is busy (allocated by another message).

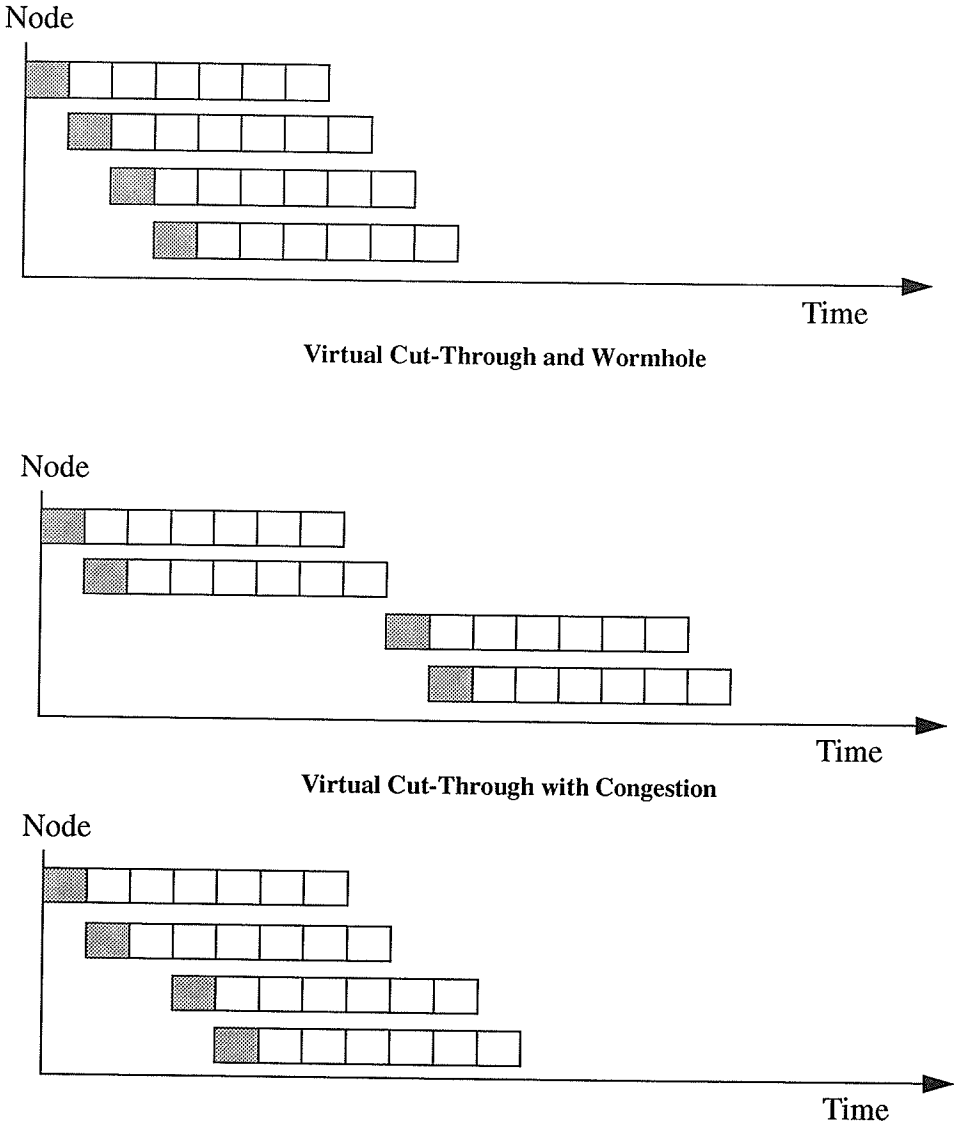


Figure 7 Comparison of Wormhole and Virtual-Cut Through Routing

The network latency for virtual cut through routing can be approximated by $(L_h/B)D + L/B$. L_h is the size of the packet header information (address destination). In most case $L_h << L$ so that the L/B term will dominate the network latency. This equation illustrates that the Virtual Cut-Through routing methodology will offer the potential for improved performance when there is no blocking in the network. Virtual Cut-Through routing will normally offer better performance in light traffic networks, and comparable performance in heavily loaded networks.

2.7.3 Wormhole Routing

Wormhole routing is new routing technique which is gaining popularity, and can be viewed as an extension to virtual cut-through routing. In this technique a packet is divided up into flits (flow control digits) which traverse the network in a pipelined fashion. The header flit contain the destination of the packet, and determines the route a message will take. Once a header flit allocates a channel, that channel is reserved for the entire packet, and no other messages will be allowed to allocate the channel until the entire packet has traversed the channel. If a header flit attempts to allocate a channel used by another packet, the flit, and the entire packet will become blocked. The packet will stop moving and will wait until the desired channel becomes free.

The network latency for wormhole routing is $(L_f/B)D + L/B$, where L_f is the size of each flit. Once again we can usually assume $L >> L_f$, and the L/B term will dominate. The advantage of this routing technique over virtual cut-through is that it does not require any buffers in the routing network. This reduces the cost of the design. The disadvantage of this technique is that it is more susceptible to deadlock.

2.8 Routing Issues

In this section we will examine some of the issues associated with routing algorithms. These issues are related to the trade-offs between cost and performance, and to the properties of the algorithm which must be ensured.

2.8.1 Performance

One of the most important characteristics of a routing algorithm is its performance. The primary measure of performance for a routing algorithm is the network latency, which is defined as the average time required for a message to be delivered to its intended destination. In general the latency will be a sum of the times required to:

1. Perform routing calculations
2. Transmit Data
3. Allocate Channels
4. Free Channels

5. Waiting Times in Queue

6. Length of Path Travelled

All of the factors, except 5 and 6, will be constant for a particular network and routing algorithm. The time spent waiting in queues will be dependent upon the traffic in the network. The length of the path between source and destination will also be a function of the traffic in a network, as adaptive routing algorithms will take alternative paths when blocked. The length of the path will also depend upon the faults which are in the network, as these will affect the paths between processors.

2.8.2 Deadlock

Deadlock is a situation where a message in the network is blocked while waiting for a resource while will never become free. In Store and Forward and Virtual Cut Through routing algorithms the critical resources will be buffers in the network. In Wormhole routing algorithms, channels will be the critical resource.

In order to ensure that deadlock is prevented, it is necessary to prevent cyclic dependencies of critical resources. It is also worthwhile to note that deadlock will occur when all the buffers in the network are full.

2.8.3 Livelock

Livelock is the situation where a message in the network is blocked waiting for a resource in the network and that resource will never be made available to that message. This is different than deadlock, where the resource is never freed. Livelock will occur when a resource is unfairly arbitrated between requesting messages.

Livelock can be avoided by ensuring that each routing function is fair, i.e. that it does not give preferential treatment to one message over another.

2.8.4 Starvation

Starvation is closely related to livelock, and refers to the situation where a message is unable to be injected into the routing network. Starvation exists when a routing function gives unfair treatment in arbitrating requests between messages in the network, and those attempting to be injected in the network. Starvation can be prevented by ensuring that all routing functions are fair with respect to messages in the network, and those being injected. As with livelock, starvation can also occur when all buffers in a network are full.

CHAPTER 3: WSI Network Modelling[1]

We are interested in studying WSI processor arrays for use in a general purpose computing environment. In this section we will introduce our models and assumptions for such networks. The models presented in this section are intended to describe these systems in a general way in order to characterize the structural and behavioural problems of WSI processor arrays while ignoring specific implementation details. We do not contend that this model is appropriate for all WSI implementations as many specific problems use alternative networks more suited for their particular application. It is hoped that insight from studying the more general purpose network will offer insights into the selection of appropriate design methodologies for specific algorithms.

3.1 Network Model

The most natural network topology for implementation on a wafer is a two dimensional mesh. Processors are arranged in a 2 dimensional array on the wafer surface. Nearest neighbour processors are connected by bidirectional communication links used for inter-processor communication. The 2 dimensional mesh network was selected as the focus of this research for the following reasons:

1. The 2 dimensional nature of the wafer surface make it naturally suited for mesh networks.

2. The detrimental effects of long wires favours topologies with nearest neighbour connections in a 2 dimensional representation.

3. The Cartesian coordinate system associated with the mesh is the simplest for the mesh network. This is important for developing efficient packet switched routing algorithms.

4. The characteristics we are investigating will be present in all networks regardless of topology. The constraints of the 2 dimensional mesh are the most severe of all suitable topologies and therefore will also be suitable for higher dimension networks.

We will assume that all processors in the network are identical (in terms of design specifications), except that each processor stores its coordinate (address) in the array. Due to difficulties in distributing global clock signals and synchronizing processors throughout the wafer, we shall assume that each processor is operating asynchronously and that delay insensitive circuits are used in communication between processors. The results of this thesis will also apply to synchronous systems.

While our physical network is a regular 2 dimensional network, the presence of faulty proc-

1. Preliminary versions of chapters 3 and 4 have appeared in [5][6][8][67]

essors leaves us with an irregular (disordered) topology. Reconfiguration techniques [62] have been proposed which allow arrays to the reconfigured to embed regular topologies, however, these techniques work best for embedding one dimensional structures. Attempts to embed two dimensional structures are usually at the expense of harvest (utilization of functional processors). We do not specifically assume any sort of reconfiguration is applied to the network. We anticipate that WSI systems will have to utilize these irregular topologies and any reconfiguration applied to the network will be used to increase connectivity between processors, not to embed regular topologies.

Let a network be represented as a directed graph G constructed from a two dimensional array of P processing elements, with m rows of n elements.

$$G = (P, C)$$

where

P is the set of processing elements $P = \{P_i | (0 \le i < mn)\}$. We will sometimes use the notation p_i to refer to processing element $P(x_i,y_i)$. $|P| = mn$. Each processing element has four possible connections from it. We designate each of these directions D, as from the set {x+,x-,y+,y-}, which correspond to positive and negative x directions, and positive and negative y directions respectively. For topologies other than the two dimensional mesh, we will augment the set with additional directions as required. We will normally use P_i (upper case) to refer to individual processors to avoid confusion with probability (denoted by small p), but may use p_i (lower case) when the meaning is not ambiguous.

C is the set of unidirectional connections between processors. We will assume that each connection has the same capacity (bandwidth), although each connection may have a different delay.

$C = \{c_i\}$. We will also use the notation $c_i(P_a,P_b)$ to indicate that connection c_i connects from P_a to P_b

In general there will be at least two connections between two adjacent processing elements (one for each direction). Additional connections may also be present, or alternatively multiple virtual channels may be implemented on the same physical connection.

The delay associated with a connection is represented by the function delay(c_i).

A path between two processors P_S and P_D is represented by Q. We define two alternative representations for Q

Let Q_P represent a path between two processing elements. We represent Q by the sequence of processing elements through which the path traverses.

$Q_P = P_1,P_2,P_3,...,P_n$

Let Q_c represent a path between two processing elements. We represent Q by the sequence of connections through which the path traverses.

$Q_c = c_1, c_2, c_3, ..., c_n$

The delay associated with a path is represented by the function delay(Q), which is defined as follows:

$delay(Q) = \sum delay(c_i)$ for all $c_i \in Q_c$

We also define two metrics on the network

$d_{min}(P_1, P_2)$ is the length of the shortest path between P_1 and P_2, and $d_{min}(P_1, P_2) = \infty$ if there is no path between P_1 and P_2. This is a measure of the logical distance between two processing elements P_1 and P_2.

$d_r(P_1, P_2) = |x_1 - x_2| + |y_1 - y_2|$ is a measure of the physical distance between processing elements P_1 and P_2.

We also define a distance function with respect to a routing algorithm R

$d_R(P_1, P_2)$ is the number of hops a message must take to be delivered from processor P_1 to P_2

Ideally we would like to implement a routing algorithm such that $d_{min}(P_1, P_2) = d_R(P_1, P_2)$, as this would be a minimal routing algorithm.

We also define an adjacency set to a processor

$A_d(P_1) = \{P_i | d_{min}(P_i, P_1) = d\}$, or $A_d(P_1)$ is the set of processing elements which have a logical distance of d from P_1.

$A_d^D(P_1) = \{P_i | d_{min}(P_i, P_1) = d, d_{min}(A_1^D(P_i), P_1) = d-1, D = \{x+, x-, y+, y-\}$, where D represents positive and negative directions in both the x and y axis. We may omit the subscript d, in which case it is assumed to be 1. This set is simply the subset of $A_d(P_1)$ whose elements are the processing elements connected in the D direction of P_1.

We will also define the following functions which will be used to indicate properties of connections and channels:

output_channels(p_i): is the set of all channels which are output channels of processing element p_i

input_channels(p_i): is the set of all channels which are directed towards processing element p_i.

source(c_i): is the processing element, p_S from which c_i connects. c_i must be a unidirectional channel.

destination(c_i): is the processing element p_D to which c_i connects. c_i must be a unidirection-

al channel.

3.2 Communication

We stated in the previous section that we are restricting ourselves to 2 dimensional mesh connected networks, but we desire to implement a general purpose processing environment. Since we wish to be able to utilize parallel algorithms which are not limited to 2 dimensional representations we must offer communications between nonadjacent processors. By implementing message passing communication protocols, and using packet switching between processors we obtain a network suitable for general purpose computation [75]. In this system, packets traverse the network by being transferred between connected processors until the packets reach their destination. Adaptive routing techniques are employed to allow packets to be efficiently routed in the event of faulty and congested processors and channels [17][57][23]. Routing algorithms are typically constrained to meet the following criteria:

1. Network latency is the delay associated with the delivery of a message. If the latency of a network is high then communication delays will be the limiting factor in performance of the network.

2. Deadlock occurs when packets become trapped due to a deadlock in the routing algorithm. Deadlock usually occurs in packet switched networks due to the finite buffer size in each processor. In order to ensure delivery messages, deadlock must be avoided.

3. Livelock occurs in a network when a packet is prevented from reaching its destination indefinitely. Like deadlock, livelock must be avoided.

In this thesis we will be using nondeterministic routing algorithms (discussed in Chapter 4) and wormhole routing algorithms (Chapter 5 and 6).

It is important to have a thorough understanding of what is meant by a communication connection between processors. Often we will draw the topology of a network and we will draw a single connection between processors. This single line can be viewed as a bi-directional connection, or as a pair of unidirectional connections. Both are equivalent as two unidirectional connections can be merged to utilize a single connection by multiplexing the opposite direction lines together. This is shown in Figure 8. In the figure, a physical connections is shown as a gray line, whereas each logical connection is shown by a solid line. The boxes labelled B represent buffers associated with each channel. In Figure 8(a) two unidirectional connections exists, each with its own set of buffers. In this case there is a one to one correspondence between physical connections and channels. In Figure 8(b), two unidirectional channels are multiplexed in a single physical connection. Each channel still maintains its own set of buffers. Finally in Figure 8(c) we show the notation we will often use in showing connectivity between nodes. We will not distinguish between physical connections and logical channels unless required. It will be important to distinguish between physical and logical connections when we consider faults on connections.

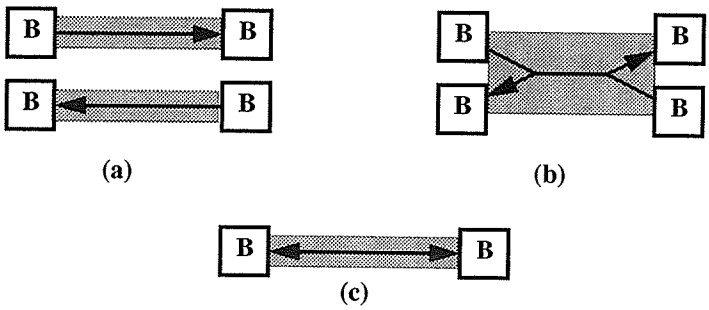


Figure 8 Equivalent Connections

It is also possible to multiplex more than two sets of unidirectional connections together along a physical connection. We can implement virtual connections or virtual channels between processing elements. This technique will be useful when we are looking at wormhole routing techniques in later chapters. Unlike the case when we are considering multiplexing two undirectional connections together, when we multiplex channels together to implement virtual channels, we must keep two buffers for each channel. This will be important in implementing deadlock free routing algorithms.

In order to avoid confusion between virtual connections and physical connections, we will use the terminology that connections refers to a physical connection between processing elements, and channel refers to a virtual channel. We may use the terminology channel, even though there is only one channel implemented using a physical connection.

3.3 Processor Model

In this section we will discuss the models that will be used for processors in the networks that will be analysed. We will be analysing three different type of message passing schemes each of which will require a slightly different model. The basic model of a processor is shown in Figure 9.

Each processor consists of two distinct entities. The Processing Unit (PU) is the computational part of a processor. It consists of the hardware which will be doing the computation in the network and contains a local memory. The Processing Element (PE) is the part of the processor which performs the communication between the PU and the neighbouring processors (PEs of other processors). We have decided to use the terminology processing element for the communication part of the processing unit so that we can distinguish between the communication part, and processing part. It still retains the processing prefix so that we associate it with the processor. It could have been alternatively called a communication element. All routing functions are implemented in the processing element (communication part) of the processor. This unit acts totally independent of the processor unit.

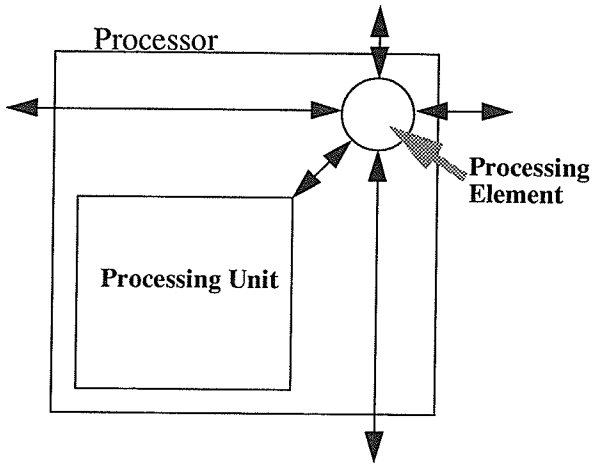


Figure 9 Processor Model

The processing unit of each processor consists of both a processor and a local memory. The processor array will most likely be fairly course grained as it would not make efficient use of resources to have a small fine grain processor with large processing element. Each processor would likely be of complexity of a small microcomputer. The specific size, complexity, and characteristics are not of direct importance to this research, as we are primarily concerned with communication in these networks. The only external connection for the processing unit is the bidirectional link to the communication unit.

Each of the processors in the network will be identical. A typical mesh connected network constructed from the processor model is shown in Figure 10.

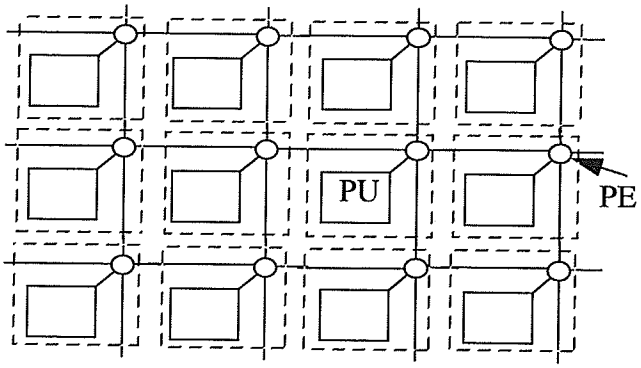


Figure 10 Processor Model Network

3.3.1 Store and Forward Processing Element

Each processor in the array is identical and consists of two separate units as shown in Figure 11. The processing unit performs all the computation and the communication unit is responsible for interprocessor communication. The routing function circuitry (R) calculates the direction for the packet to be routed at the processor. It may utilize information for the current state of the processing element, local channel and queue utilization,.

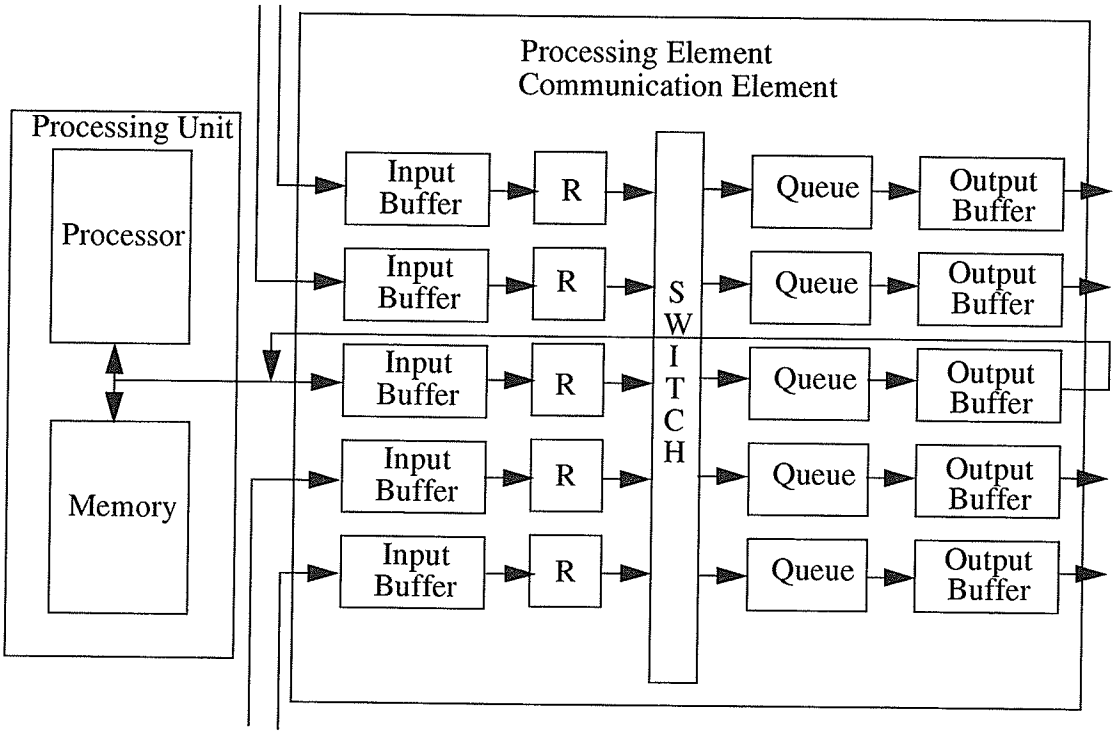


Figure 11 Store and Forward Processor

The communication unit consists of five input and five output buffers, five FIFO (First In First Out) buffers, and packet routers. Upon receipt of a packet from a neighbouring processor or processing unit in one of the input buffers, the packet is directed to a routing circuit which determines which output buffer the packet will be placed in. The packet is next transferred through a crossbar switch to the queue of the appropriate output buffer. Communication to and from the processing unit is handled in a consistent manner to that of neighbouring processors.

This model of the store and forward routing processor is suitable for static and oblivious routing algorithms only. The determination of which output buffer will be used is determined solely by the destination address of the packet. Once the output buffer appropriate for the message is selected, there is no opportunity available by which the message can choose an alternative output buffer.

An alternative implementation of a store and forward routing processor in shown in Figure 12. In this model, only a single queue is used to store buffered messages, and the routing function is not applied until the message exists from the queue. This implementation has the advantage over the previous model in that the routing function can more easily utilize information about the status of the output queues, and is thus more suitable for implementing adaptive routing functions. A

feedback path from the routing function to the message queue allows an opportunity for messages to have the routing function reapplied. This will be an important feature in the non deterministic routing algorithms which will be discussed in chapter 4. As shown this processor will only route packets one at a time, and is hence sub-optimal. This model could easily be extended to allow multiple packets to be routed simultaneously.

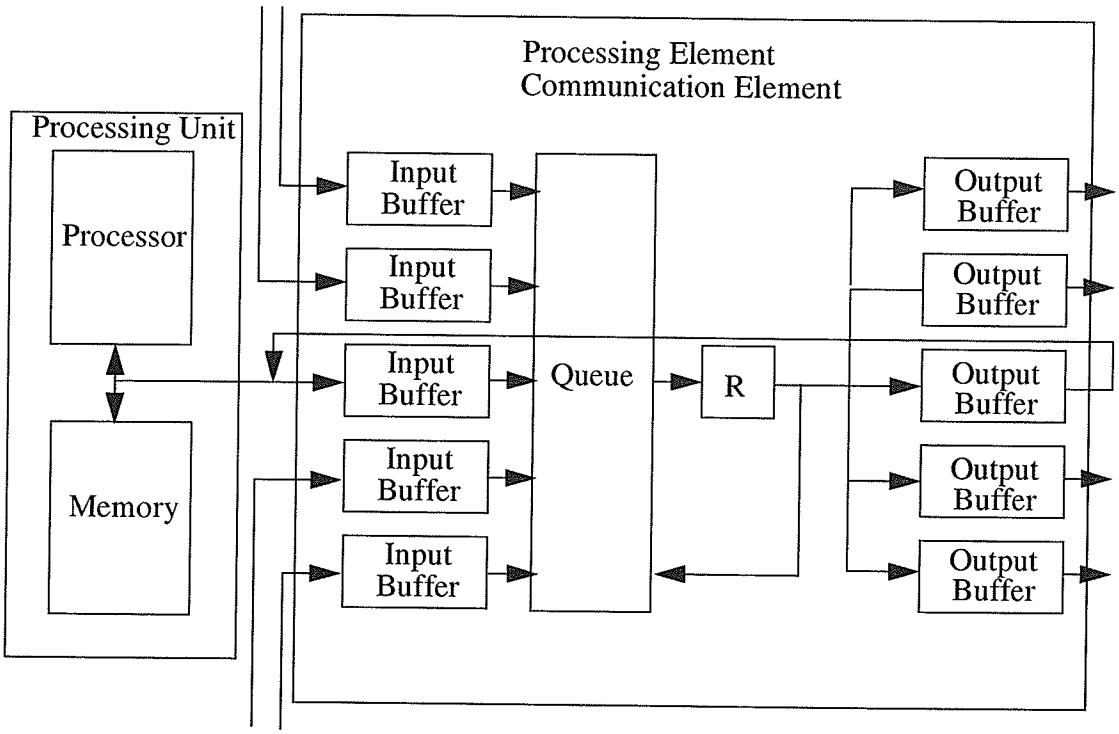


Figure 12 Alternative Store and Forward Processor

3.3.2Virtual Cut-Through Processing Element.

The communication element of the Virtual Cut Through processor is different from that of the store and forward processor. Whereas the store and forward processor will store each message in a queue, the virtual cut through processor will only do so when a message is blocked. If a message is not blocked, the message can be transferred through the processor without any buffering. A model of the virtual cut through processor is shown in Figure 13.

Once a packet is received by the input buffer, it is passed through a routing circuit which will select an output buffer through which the message will traverse. If the selected output buffer is not busy (allocated by another message), the packet will be sent to the selected output buffer. If the output buffer is busy, the message will be buffered into the message queue. Here the message will remain until the entire packet is received. Once the entire packet is received, the message may

then attempt to select an output buffer. Once an output buffer is available, the message will be transferred to the neighbouring processor.

Although only a single buffer is shown in this model, it is possible to utilize multiple queues, one for each output buffer, and thus the need for recalculating the routing function will be avoided. This is similar to the model used for the store and forward processor.

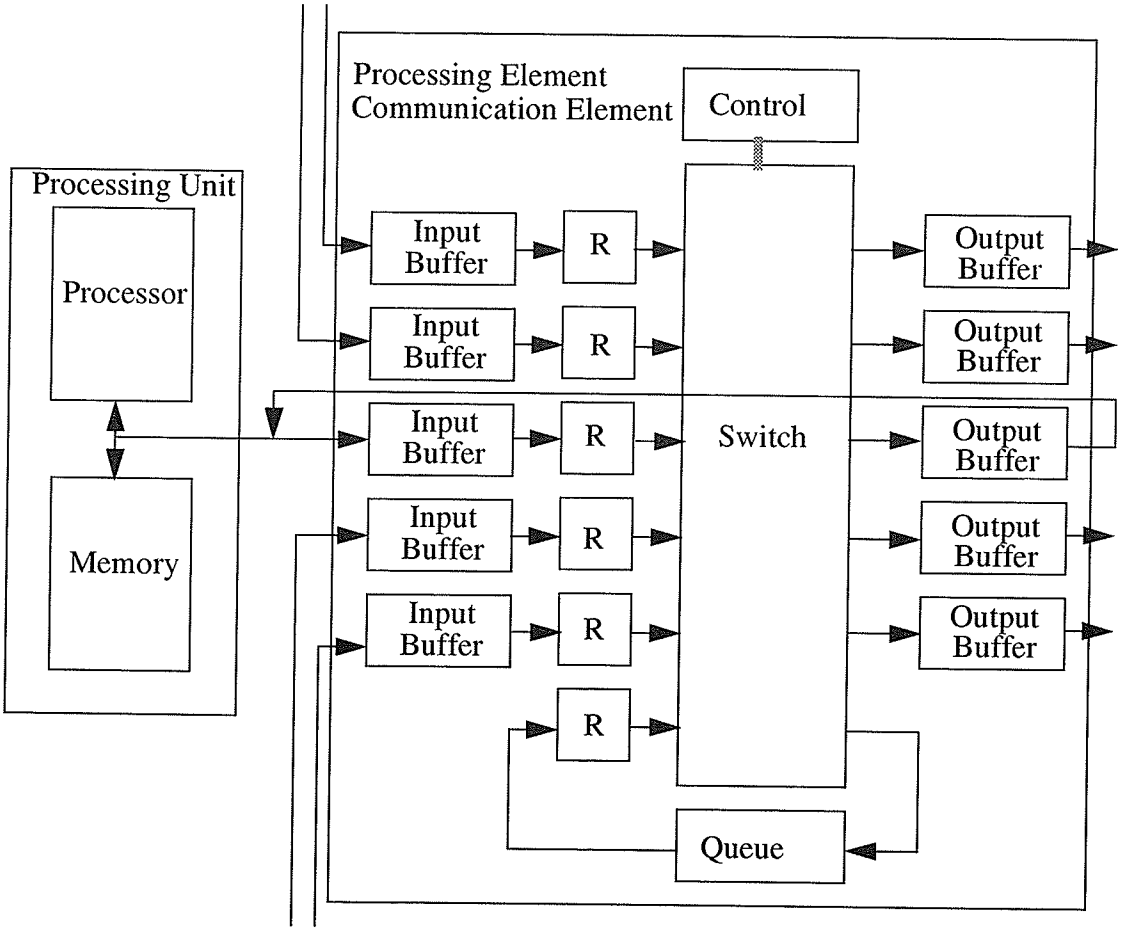


Figure 13 Virtual Cut Through Processor

3.3.3 Wormhole Processing Element.

A model for a wormhole routing processor is shown in Figure 14. This model is almost identical to that of the virtual cut through processor, except the queues for storing messages have been eliminated. This offers the advantage of a much simpler and smaller routing unit as no buffers are required.

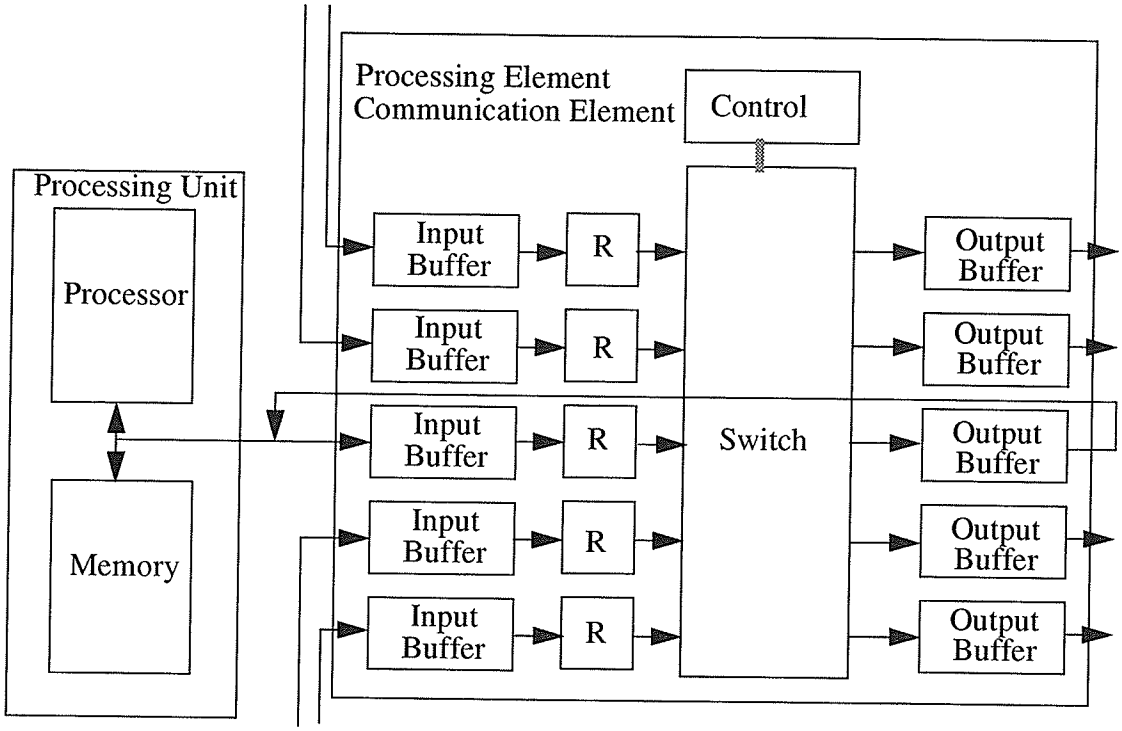


Figure 14 Wormhole Routing Processor

We will make the following assumptions based on [19] but with restrictions preventing adaptive routing and nonconnected routing algorithms removed:

1. A message which arrives at its destination is eventually consumed. Without this assumption, the network may become congested with messages which have arrived at their intended destination, but are still in the routing network. This also assumes that the processing unit is always capable of accepting a message. We are neglecting consequences of programming errors causing deadlock in the network.

2. Each processing unit can generate messages destined for a subset of the processing units in the network. This does not have to be a proper subset. We will naturally try to have each processor be able to communicate with each other, but as we will see later, with faulty processors this may not be easy.

3. Each processing unit can receive messages from any other processing unit

4. Each processing unit can generate a message of arbitrary length.

5. Once a queue accepts the first (header) flit of a message, it can not accept any flits from other messages until the entire message has been accepted.

6. An available queue may arbitrate between messages that request the queue space, but may not choose among waiting messages.

7. Nodes can produce messages at any rate subject to the constraint of available queue space (source queued).

3.4 Fault Model

One of the hurdles in utilizing WSI processor arrays is the inherent existence of faults on the wafer. We assume that all processing elements will be designed with fault tolerant techniques in an attempt to improve yield. However, it is still unlikely that all faults may be tolerated. As a result we assume that a certain percentage of processors in the network will be faulty (non operational). At the same time we assume that no faults will occur in the interconnections between processors. This is usually justified since the interconnects contains no active components and are significantly smaller in area than the processing units. Interconnect faults (assuming they are detectable) will not prevent the network from operating, only degrade its performance.

Unless specifically stated, we assume an independent fault model for processors. Each processor is assumed to be functional (faulty) with probability p (1 - p), independent of the state of neighbouring processors.

In order for WSI processor arrays to operate in the presence of faults, it is imperative that each faulty processor's neighbours be aware of its faulty state. Identification of faulty processors may be achieved through the use of self-test, boundary scan (JTAG)[36], or consensus techniques. Once a faulty processor is identified, it must be isolated from the network, either logically or physically depending upon the fault characteristics.

We define two subsets P_f and P_o of P such that

P_f is the set of faulty processing elements and P_o is the set of operational processing elements

$P = P_f \cup P_o$ and $P_f \cap P_o = \Phi$

Similarly we define two subsets C_f and C_o of C such that

C_f is the set of faulty connections and C_o is the set of operational (fault free) connections

$C = C_f \cup C_o$ and $C_f \cap C_o = \Phi$

In order to utilize this fault model, we must be able to determine the location of the faults in the network. This can be accomplished by using DFT and BIST methodologies in the design of the PE and PU. We will not be looking at faults which cause shorts between power and ground, and assume that some sort of isolation circuitry exists to isolate fault elements from the network so that

they do not interfere with the performance or characteristics of non faulty elements.

We will be only looking at static faults in this thesis. We will assume that testing is performed on the network prior to operation of the network, and the location of faults is known. It is possible to attempt to provide fault tolerance to dynamic faults, but this would require a sophisticated monitor and roll-back recovery system which is beyond the scope of this thesis. The algorithms and analysis presented in this thesis do not necessarily preclude dynamic fault tolerance, but simply do not attempt to handle all the problems associated with that fault model.

3.5 Routing Algorithms

Traditionally, routing algorithms have been classified as either deterministic or adaptive. The distinction between the types is whether the path between source and destination can be determined uniquely from the source and destination address, or whether the path depends upon static and dynamic conditions in the network. We introduce a third category here to allow for nondeterministic algorithms.

Definition 1: A routing algorithm is classified as deterministic if the path traversed by a message is uniquely determined by the source and destination addresses. This type of algorithm is often called oblivious.

Definition 2: A routing algorithm is classified as non deterministic if the path traversed by a message is determined solely by the source and the destination addresses and a set of random values. The set of random values may be bounded or unbounded.

Definition 3: A routing algorithm is classified as adaptive, if the path traversed by a message is dependent upon network conditions.

An adaptive routing algorithm may be either deterministic or non deterministic.

It is also important to distinguish between routing algorithms which are independent of location, and those which are dependent upon the location in the network. For example, in a routing algorithm such as XY routing, the routing at any particular node in the network is identical, whereas in other algorithms (which will be examined later in this thesis), the algorithm will be dependent upon the physical location in the network.

Definition 4: A routing algorithm is said to be homogeneous on a network, if the routing algorithm is not dependent upon the location in the network. A routing algorithm which is not homogeneous is called heterogeneous.

Deterministic algorithms are primarily used in networks with very little congestion, and no faults. Almost all routing algorithms which are of interest to us will be adaptive. In an adaptive routing algorithm, each processing element will make a decision on which channel to route a message. We define this decision as follows:

Definition 5: A routing function R is a mapping from input connections to out-

put connections.

Since an adaptive routing function has to make decisions on how to route a message based on network information, we define the following classifications of routing algorithms, to indicate how much information is available to each processing element.

Definition 6: A routing function is k-local if it utilizes only information for processing elements which are a distance of k or less from the current processing element. A 1-local routing functions is also called a local routing function.

Definition 7: A routing function is global if it utilizes global network information in selecting a path. This can also be thought of as a d-local routing algorithm where d is the diameter of the network.

When we are dealing with disordered arrays, it will often be impossible to implement a local routing algorithm on the network. It is not feasible to implement a routing table for each processor in the network, as this would require an unrealistic amount of storage as the network size grows. An alternative approach will be to use non-local information, but only a constant amount, not dependent upon network size.

Definition 8: A routing function is quasi-local, if it utilizes non local information in its routing function, but the amount of information is dependent upon the topology, and not the network size.

Recall that when a processing element was described, it was defined to contain knowledge of its own state and the state of all connections attached to it. The state of each connection also reflects the state of each processing element which it connects. Thus each processing element knows its state, the state of all its connections, and the state of all processing elements connected to it through operational connections.

An important property of any routing algorithm is that it must be able to route between all possible pairs of source and destination addresses. In subsequent chapters we will be looking at routing in networks which contain both faulty processing elements and connections. We now define the following terms which will be used to distinguish routing algorithms which can route between all addresses, and those which can not.

Definition 9: A routing algorithm is said to be complete or connected with respect to a fault set, if it can successfully route messages between any two processors in the network.

Definition 10: A routing algorithm which is not complete is said to be not connected.

We will also use the concept of a routing permutation when discussing routing algorithms. Multistage Interconnect Networks (MINs), which are indirect networks, often establish connections between inputs and outputs of a network. Because of the limited resources of these networks not all inputs can connect to all outputs simultaneously. Consider a MIN with m inputs and m out-

puts. A permutation is a set of m distinct addresses applied to the inputs of a network. A permutation can be routed if all the inputs can simultaneously connect to their designated outputs.

Another distinction which will be significant between different routing strategies will be when the routing function is applied to messages. In a static routing function, there is only one predetermined path by which a message traverses the network. When a message is received at a processing element, the routing function may be applied, and the message may wait in the appropriate queue for an output channel. In the case of adaptive routing functions, if a message selects an output channel, and then waits in its queue until it is available, it may be waiting in a queue for a channel while another channel becomes available. In cases like this it is often advantageous to recalculate the function multiple times, until the message is accepted along one of the available channels. In order to distinguish between these cases, we will define the following types of routing functions:

Definition 11: A routing function is said to be singular if the routing function if applied only once to an incoming message.

Definition 12: A routing function is said to be repeated if the routing function is repeatedly applied to waiting messages.

In the case of repeated routing functions, it is important to define some sort of rate at which the routing function will be reapplied. Since we are only looking at asynchronous systems in this thesis, it is not possible to have the network recalculate the routing function at regular synchronized intervals. Each processing element will be operating from an independent clock.

Definition 13: A routing function application function is a function associated with repeated routing functions, which indicate at which times the routing function will be recalculated.

CHAPTER 4: Nondeterministic Adaptive Routing[1]

In this chapter we will examine the properties of a processor array in the presence of faulty elements. We will utilize percolation theory to determine bounds on the number of functional connected processors we can expect to obtain as a function of yield. We will also use percolation theory to establish characteristics of message flow in these networks. Insight from this studies will be used to develop a new routing technique for disordered processor arrays using nondeterministic routing techniques.

The non reconfigured, faulty mesh connected network which we will be examining in this chapter is one of the most hostile environments in which to attempt to implement a message passing system. We will attempt to take a different approach to utilizing this environment from the more traditional approach of reconfiguring the network to produce a regular topology. We will attempt to exploit the disordered nature of the faulty processor array, and will implement a routing algorithm which works in this environment.

4.1 Percolation Theory, Anomalous Transport and WSI

Percolation theory [32][77] provides a general platform for analysing binary mixtures. Large systems may be modelled using site percolation theory as a lattice with each vertex in one of two states. Alternatively, bond percolation theory can be used by modelling each edge in one of two states. Examination of the characteristics of clusters of connected nodes of similar states shows unexpected complex behaviour including singularities during phase transitions in these simple systems. Percolation theory has been used to characterize phenomena observed in physical systems including annealing, semiconductor devices, and VLSI arrays [28][61]. The application of percolation theory to WSI processor arrays is achieved by modelling elements in the array as either in a functional or faulty state. Bond percolation may be used to model systems with faulty interconnects between processing elements, and similarly site percolation may be used to characterize systems with faulty processors.

In analysing WSI processor arrays we are primarily interested in the size and distribution of connected clusters of functional processors, especially the largest cluster. Typically the characteristics we are concerned with may only be found through simulation, as analytic solutions are only known for 1 dimensional and simple 2 dimensional cases [73][77]. In studying systems using percolation theory, we look at systems at which the state of each node in one of two states. The probability that a node is in one state is specified by the probability value p (the probability that the node is in the other state is 1-p). We assume that each node's state is independent of the state of other nodes. Two of the most interesting parameters of direct interest to us here are the percolation threshold p_c and the percolation probability $P(p)$. The percolation threshold p_c is the value of p (p

1. Preliminary versions of chapters 3 and 4 have appeared in [5][6][8][67]

here is analogous to yield in a WSI system) at which point the system undergoes a phase transition. In an infinite system p_c is the value of p for which an infinite cluster exists for all values of $p \geq p_c$. In a finite WSI system p_c refers to the formation of a spanning cluster, a connected cluster of working processors which spans the entire wafer. The spanning cluster represents the largest connected component of the network, and for systems above the percolation threshold, this component will span the entire network. Not all functional processors will be in the spanning cluster. The percolation probability $P(p)$ represents the probability that a randomly selected site is part of the infinite or spanning cluster. For a WSI processor array $P(p)$ represents the probability that a processor site selected at random is part of the collection of working processors available for collective computation. Figure 15 illustrates the behaviour of $P(p)$ for a large 2-d square lattice. For $p < (1\text{-}p_c)$ the faulty processors in the array will form a spanning cluster. At this yield level, the function processors will only form a very small clusters of processors. Clearly, unless we utilize some form of reconfiguration, networks with $p < p_c$ will not be useable.

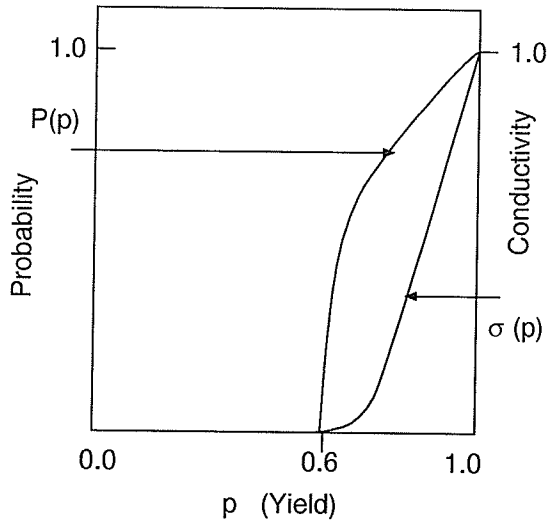


Figure 15 Mesh Percolation Parameters

Also shown in Figure 15 is the parameter σ denoting the conductivity. This value is calculated is an analogous manner to calculating the conductivity of a random resistor network[77]. Conductivity is used as a measure of the quality of paths through the network. Even though a spanning cluster exists for $p \geq p_c$, the conductivity of the network is less than 1 for $p < 1$ and degrades appreciably as p approaches p_c. For WSI networks the conductivity is used a measure of the quality of communication paths through a network. A conductivity less than 1 indicates that packets traversing through the network will be hampered by the presence of faulty processors.

An example of a network above and below the percolation threshold is shown in Figure 16 and Figure 17 respectively. A 50 by 50 array is shown with the largest cluster of functional processors emphasized. This figure represents one possible configuration of the network for this value of p. Faulty processors are not shown. Below the threshold the largest cluster does not span the en-

tire array. Smaller clusters of functional processors also exists in the array but would not be accessible without some form of reconfiguration. These figures also illustrate the degree and type of local interconnection (e.g. nearest neighbour) that would be required to connect up all working processors for a given yield. In the array with $p > p_c$ (Figure 17) a spanning cluster now exists, and as predicted from Figure 15 not all functional processors are in the spanning cluster. It is also worthwhile to exam the connectivity of nodes in the array. Clearly it may be seen that nodes on alternative sides of the array are connected through a small common set of nodes in the array (e.g. removal of two nodes can partition the array into two clusters). This derives from the observation that we are near a phase transition where the state of a small number of processors determines whether the spanning cluster exists. These common nodes (including the nodes which change states between Figure 16 and Figure 17) represents places where bottlenecks will form in the communication across the array. This is reflected in the conductivity which is only slightly greater than zero for the case shown in Figure 17.

A node is referred to as an articulation point if its removal will cause its cluster to become separated. Similarly any connection whose removal will cause the cluster to be separated is referred to as a bridge. It is the existence of articulation points and bridges which will be hot spots during routing.

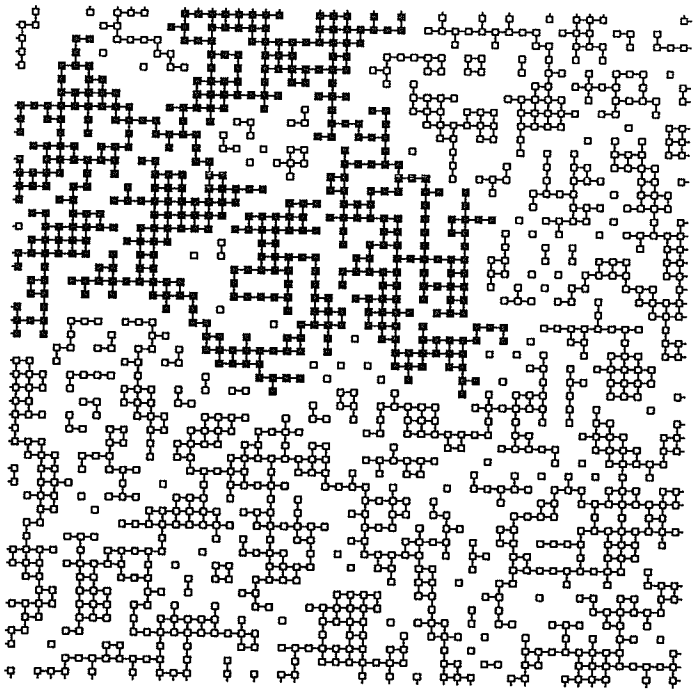


Figure 16 Network with p = 0.57

It has been noted that the infinite or spanning cluster demonstrates fractal behaviour [72]. The effects of the fractal geometry are directly manifested in transport studies [11] which indicate that transport or message passing on a fractal network can look "anomalous", analogous to transport observed in disordered semiconductors. This anomalous transport has been modelled as a non-Gaussian, non-Markovian transport model by Scher and Montrol [74]. The main result is that the transit time taken by a message can be significantly greater than one would normally associate with Gaussian transport phenomena. Although not thoroughly investigated, the extent this fractal characteristic is evident within 10-15% of the percolation threshold [73]. The WSI consequence is that, message passing on the array will be significantly degraded for yield values near p_c. This performance degradation is strictly a result of spatial disorder. At higher values of p and in fact for yield values of 100% anomalous transport behaviour can still be observed if the array has sufficient temporal disorder. This is in fact the first anomalous transport model introduced by Scher and Montrol. As such, for WSI message passing environments with sufficient congestion or traffic, the message passing performance can also appear anomalous.

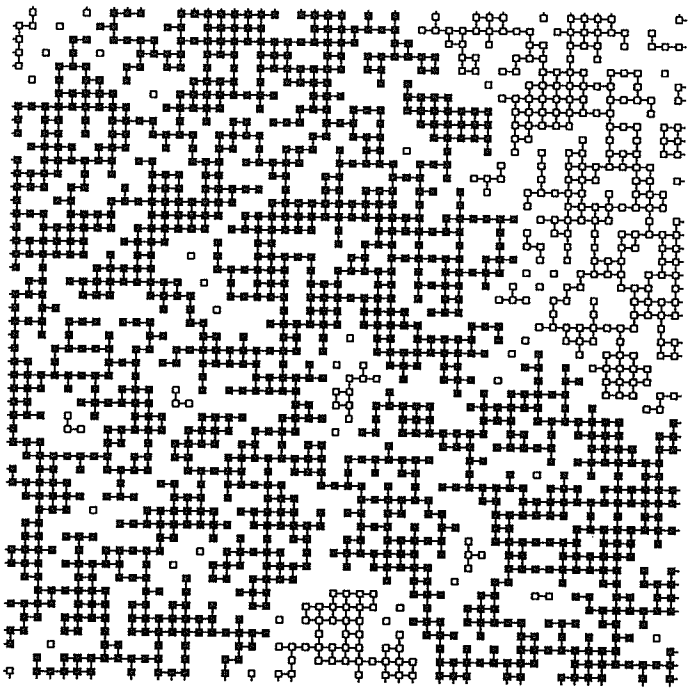


Figure 17 Network with p = 0.63

Parameters such as p_c are directly affected by topology. Table 1 illustrates p_c for various

dimensions and lattices[77] . The values for the hypercube are listed as ? since the values will de-

Topology	p_c (site)	p_c (bond)	Vertex Degree
2-D Mesh	0.59	0.5	4
Honeycomb	0.70	0.65	3
Hexagonal	0.5	0.35	6
3-D Mesh	0.31	0.25	6
Hypercube	?	?	?

Table 1: Percolation Thresholds

pend upon vertex size (and thus network size). As the number of nodes in the system increases, the vertex degree of each processor will increase and the percolation threshold will decrease.

For our discussion purposes, we are considering only nearest neighbour 2-dimensional arrangements. Alternative structures such as binary hypercubes are not considered as they are not conducive to WSI layout or implementation. As such, for the remainder we will be primarily concerned with mesh topologies.

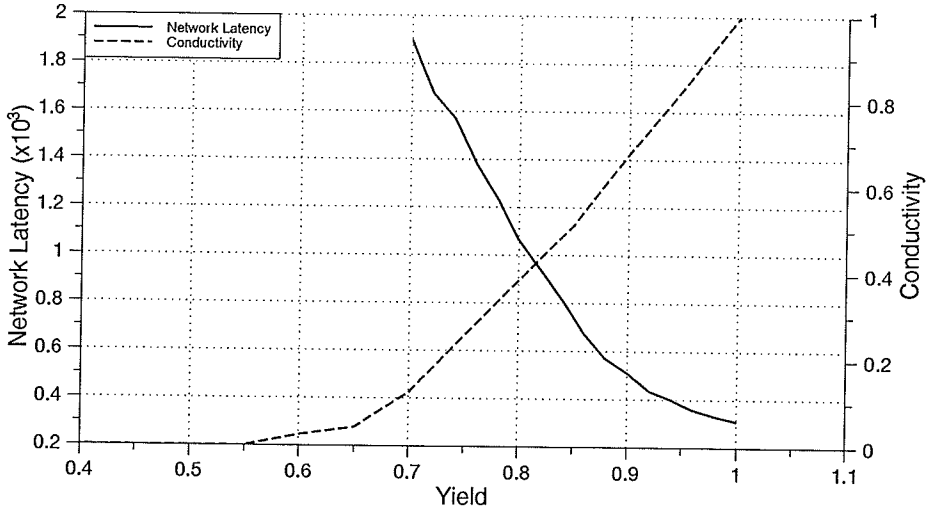


Figure 18 Percolation Characteristics

With the mesh-connected networks the yield must be relatively high, and in fact significantly higher than p_c. Based upon simulated transport studies the wafer yield should ideally be above 90% [5][6]. As such, in order to achieve an acceptable yield some type of reconfiguration may be necessary. For example, interstitial redundancy may be used within local blocks in order

to effectively improve the yield [76]. Alternatively, this may be viewed as decreasing p_c by increasing the average vertex degree of the processing elements. It should be noted that the objective of this type of preprocessing reconfiguration is not motivated by an attempt to embed a specific architecture or algorithm, rather it is viewed as a means of pushing the array into a realm of acceptable yield for more general purpose computation. In this light we are more interested in reconfiguration to obtain a sufficient degree of connectivity as opposed to the standard definition of yield. In the sequel we will assume that this type of reconfiguration if required would have been completed; this allows us to focus on a mesh architecture without loss of generality.

Figure 18 shows results of simulations of a routing algorithm on a disordered array with different yield. One graph shows the conductivity of the array, and the second graph shows the latency of the network. These two graphs are inversely proportional as we would expect from percolation theory.

Percolation theory does not directly give us any formal methods which we will help us implement routing algorithms for disordered systems, but the theory does provide us with insight into the characteristics and bounds on what we can reasonable expect to accomplish. The first important observation of from these systems is that if the system is not above the percolation threshold, we will be unable to utilize the functional processors as they will be isolated. Reconfiguration will be mandatory for these systems. For systems with yield rates above the percolation threshold, we will have a large connected network of functional processors, but will still have low bandwidth between connected processors.

4.2 Adaptive Nondeterministic Routing

In the previous chapter we discussed some of the characteristics of the physical environment available for use in a WSI processor array. We now direct our attention to the problem of routing algorithms for this environment. In an ideal network, all packets should normally travel along one of the shortest paths from source to destination to ensure the best possible performance. In WSI networks, or any irregular topology, the shortest path between two processors is not always known to the processors in the network. Adaptive routing techniques are used to ensure delivery of packets in the presence of faulty processors and connections.

Although numerous adaptive routing algorithms have been investigated [23][63] most are not designed to be used in a wafer scale environment. Based on percolation theory discussed in the previous section, there are four possible scenarios for yield rates in a WSI processor array:

1. $p >> p_c$. This is the case where we either assume a very high yield rate ($p > 0.95$) or a topology such that p_c is very small (Hypercubes of dimension greater than 10 have $p_c < 0.1$ [61]). Most routing algorithms fit into this category. Two dimensional mesh routing algorithms normally assume fault free processors and interconnects [34]. Routing algorithms designed for faulty environment normally require a topology which is highly connected, such as a hypercube [17] [47] or other structures including k-ary n-cubes [57]. In both these cases it is usually possible to implement shortest path (minimal) routing algorithms

2. $1 > p > p_c$. Although a spanning cluster exists in this region, and most functional processors are connected, the conductivity of the network is considerably less than 1. In this region the network will typically consist of clusters of functional processors connected to each other through a relatively small number of connections. In order to route messages in this environment a message must be able to find a path to its destination which bypasses faulty processors. Since this route can not be determined from local information at each point in the network, a message must be able to explore alternative routes and back track when it encounters dead-ends.
3. $p \approx p_c$. Percolation theory tells us, and simulation confirms [5], that routing in this environment is not practical. Although a spanning cluster exists, the conductivity is close to zero. Messages will become congested at processors which connect functional clusters. Only networks injecting a small number of messages will be able to utilize such a network.
4. $p < p_c$. Since most functional processors exists as isolated clusters unable to communicate with each other, there is no effective way to utilize such a network.

Deterministic routing algorithms in WSI processor arrays generally require either non local processor status information be available or history information be associated with each packet indicating the path which a packet has travelled to reach the current processor. Nondeterministic algorithms offer the advantage of not requiring such information. Unfortunately, the behaviour of these algorithms is not easily determined through analytic means; as a result, we must judge algorithm performance based on expected delivery times, and use higher level protocols to handle packet non delivery. At the routing level it is imperative that the routing algorithms attempt to route messages such that not only are the expected delivery times reasonable, but the number of undelivered (trapped) messages is minimized.

4.3 Nondeterministic Routing

In this study we shall restrict ourselves to the examination of the behaviour of these algorithms on a simple 2-dimensional mesh topology. Each processor is connected to each of its nearest neighbours. This simple architecture was chosen in an attempt to most accurately model the layout structures of most WSI configurations. A portion of a typical array is shown in Figure 19. The behaviour reported in this study will also reflect the behaviour of other topologies. This topology is also useful in the analysis of other processing systems, including distributed computing networks.

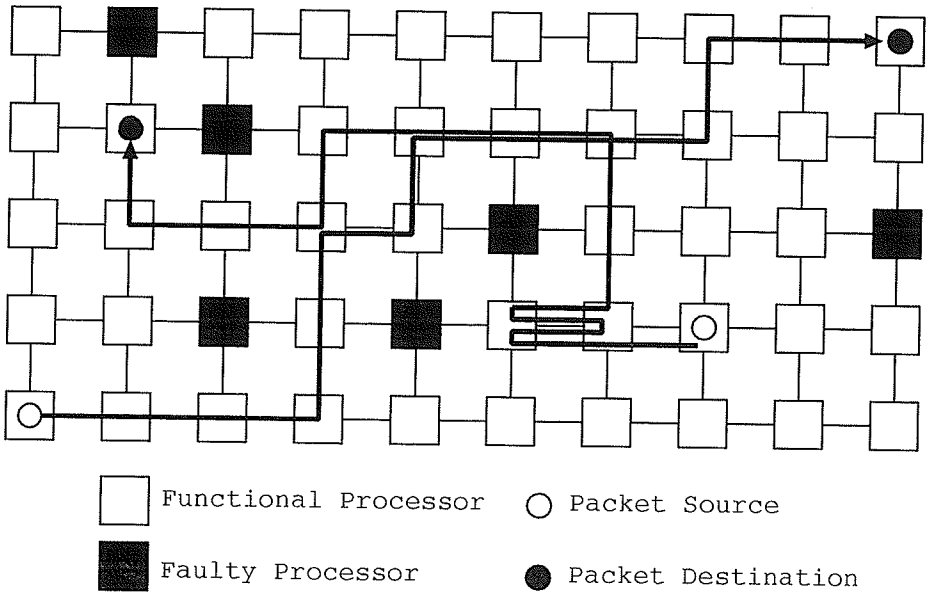


Figure 19 Nondeterministic Routing

Communication in the network is implemented by a message passing technique. Each message M can be described by three parameters, $M = (P_S, P_D, S)$, where P_S is the source processor of the message, P_D is the destination processor of the message, S is the size (number of packets) of the message. We will normally assume that each message contains a single packet. Although we do not deal explicitly with multiple packet messages, techniques such as wormhole routing may be adapted to these algorithms.

In the simplest nondeterministic routing algorithm, a set of probabilities associated with the network describes the probability of a message being routed on each of the processor's connections relative to the packet's destination. These probabilities are referred to as the routing bias values (B_R) for the network, and are represented as follows:

$$B_R = (p_f, p_l, p_b, p_r)$$

such that

$$p_f + p_l + p_b + p_r = 1$$

$$pf > pl, pr, pb$$

where p_f represents the probability of a message being routed towards its destination, p_l to the left of its destination, p_r to the right of its destination, or away from the destination p_b. The first equation asserts the criteria that a packet must be routed down one of the four connections as the four biases equal one. The second equation asserts the criteria each packet is biased towards its desti-

nation, as the forward bias is greater than all the other biases. This implies that the biases favour heading toward the final destination. Without this criteria, messages would probably be routed away from the destination.

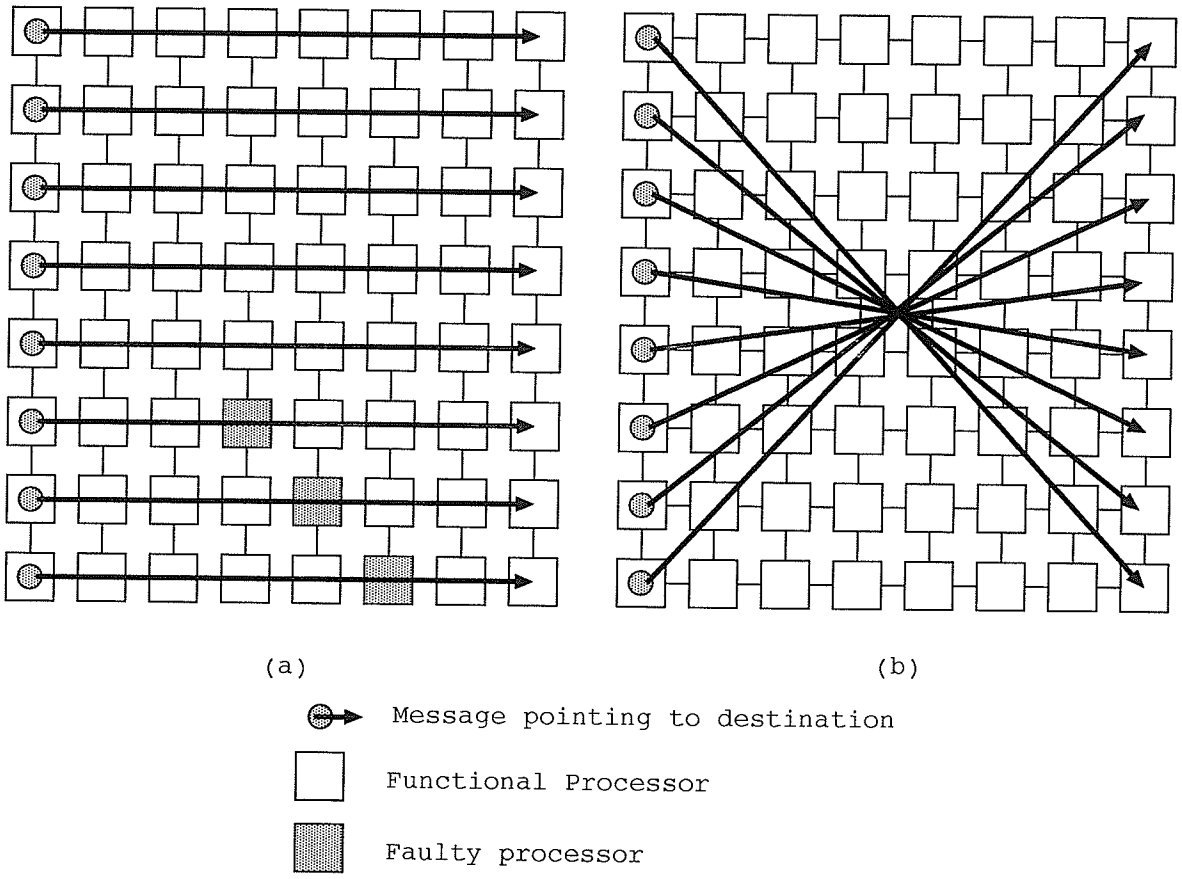


Figure 20 Generic Problems in Routing

A packet traverses the network by propagating through a sequence of processors until it reaches its destination. The route is determined stochastically as the packet progresses towards the destination based upon the global bias of the network. Figure 19 illustrates the movement of two packets in the array. The path that each packet takes is indicated by the thick line. The packet travelling right to left illustrates two important aspects of the network. First, the packets do not always choose the best path, and may have to back track to reach their destination. Secondly, even if a packet backtracks, it may once again take a previously taken dead end path. The second packet, which traverses left to right, shows that packets may be on collision course with other packets. The movement of a packet as it traverses the network is analogous to the biased random walker [50].

This type of routing algorithm can be shown to be deadlock free. We know that once a message is in the routing network, it can always select any of the neighbouring processors as a destination. The routing algorithm is a repeated one, so that the destination is recalculated as a message is stored and blocked. As a result there is always a finite probability that a message will be trans-

ferred to each of the neighbouring processors. The case where deadlock could arise, is when all buffers in the network are full. Deadlock can be avoided in this situation by allowing a mechanism which allows messages to be exchanged between two processors without requiring additional buffer space. As we shall see when the network is saturated like this, message transport will be very slow, and it will be sufficient to ensure message movement, and hence there exists no deadlock.

Livelock and starvation will not be possible since all arbitration among queues and requests is done fairly, with no priority given to particular messages.

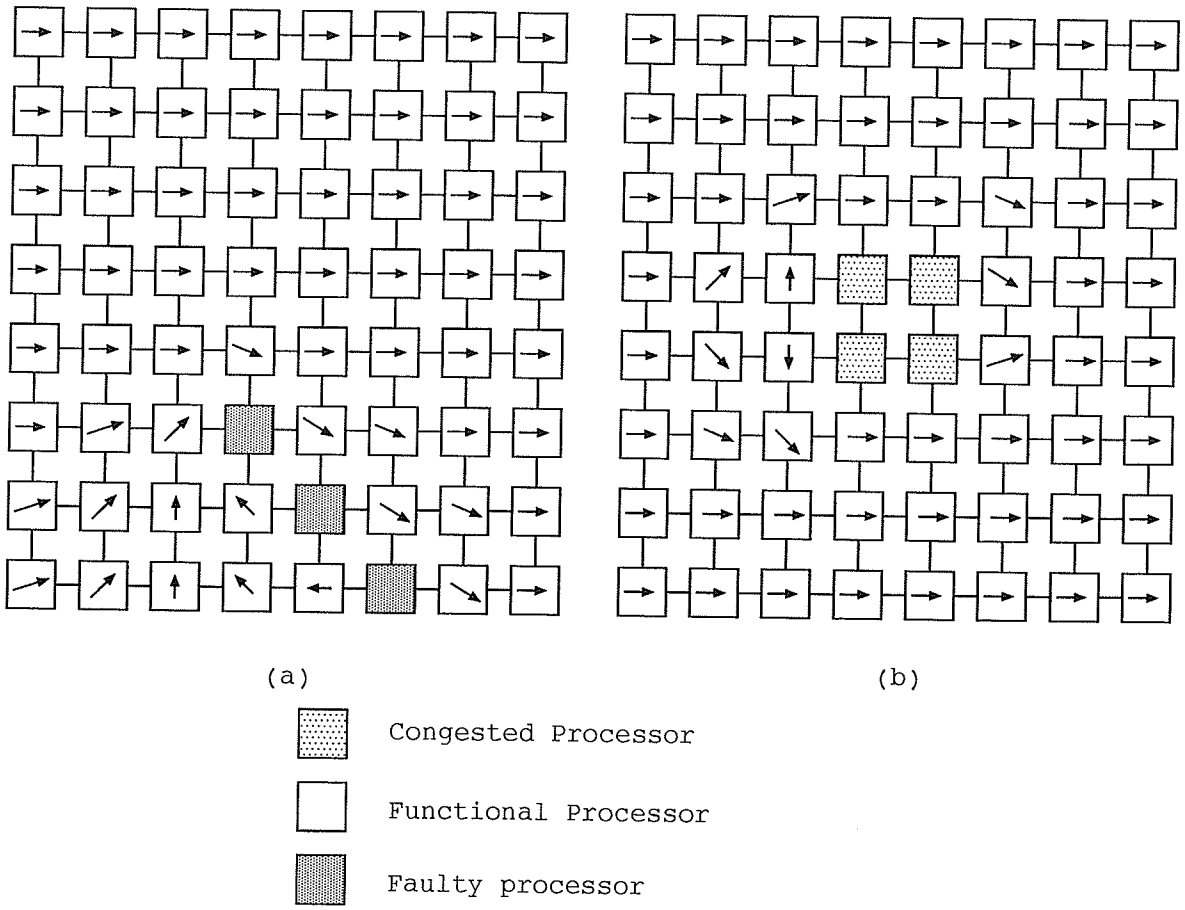


Figure 21 Biases from Faults and Congestion

4.4 Adaptive Bias

Simulations of packet switched networks revealed that the poor performance of faulty processor arrays was due primarily to two factors [5]. First, some messages were becoming trapped at clusters of faulty processors. In Figure 20 (a), packets would become trapped near the bottom of the array. Once a packet encounters the faulty processors, it has to back up and traverse around the faulty cluster. With a strong forward bias, this backtracking becomes a long procedure. Secondly, the center of the processor array through which most messages travelled became congested as il-

lustrated in Figure 20 (b). The faulty processors also formed clusters in scattered locations which created congested locations which packets must flow through. The presence of these congested locations significantly degraded performance. Based on these qualitative observations an adaptive biasing technique was devised allowing the bias values of a network to adapt to faulty processors and congestion. The goal is not to use a single set of biases for the entire array, but instead to adapt the biases locally at each processors in response to local conditions such as the presence of faults and congestion. Figure 21 (a)(b) shows how the bias values should direct a packet which is traversing the network from left to right. The arrows show the favoured direction of the local biases. Since the biases are relative to the destination of the packets, these arrows represent only the biases for packets traversing left to right. A routing algorithm using this non local information is referred to as quasi-local.

In order to modify the bias of a network to reflect the faults and congestion of a network, consider the bias as a sum of three different tuples, representing the different components of the bias. The bias $B^i(t)$ which is the bias values at time t for processor i is defined as follows:

$$B^i(t) = a_0 B_R + a_1 B_F{}^i + a_2 B_T{}^i(t)$$

where:

- B_R is the network routing bias previously discussed.
- B_F is a set of bias values reflecting the faults in the network.
- $B_T{}^i(t)$ is a set of bias values representing the congestion of messages in the network.
- a_j, $j \in \{0,1,2\}$ is a coefficient used to weight the different components

The determination of the B_F term is based on an iterative learning algorithm similar to the delta rule used in artificial neural networks [58]. During learning each processor updates its fault bias terms based on the values of neighbouring processors. The transfer of bias values can be accomplished by either passing the values in packets or through the use of special interconnects between processors. Alternatively, the bias value may be calculated off line and loaded into the processors prior to network operation. Each method works equally well as only nearest neighbour information needs to be exchanged. The updating of bias values is based upon the following equations:

$$B_iF(tl) = \widehat{B}^i{}_F(t_l) / | \widehat{B}^i{}_F(t_l)|$$

$$\widehat{B}^i{}_F(t_l) = B^i{}_F(t_l - 1) + \alpha_F \Sigma^k (B^k{}_F(t_l - 1) - B^i{}_F(t_l - 1))$$

where

- t_l is the time during training of fault bias.

- k is the index of a neighbouring processor
- α_F is a learning rate. $0 < \alpha_F < 1$.
- $\widehat{B}^i_F(t_l)$ are the new non normalized fault bias values.
- $B^i_F(t_l)$ is the new fault bias values, normalized such that $p_f + p_l + p_b + p_r = 1$.

Initial condition for the learning algorithm are as follows:

$B^i_F =$

(0.25,0.25,0.25,0.25) if i is a faulty processor

(0.0,0.0,0.0,0.0) if i is a functional processor

Although faulty processors are unable to transfer their bias values, since each processor knows which of its neighbours are faulty it does not need to obtain values from the faulty processor and can use the initial conditions for the faulty processor.

We assumed the fault biases are learned prior to network operation and the values remain constant during normal network operation. It is possible to use the same approach to dynamically adjust the bias values to adapt to failures in processors occurring during circuit operation. Mechanisms to redirect packets already in the network to replacement processors would have to be implemented which is beyond the scope of this study.

The bias adjustment for congestion B_T is used to change bias values around areas where packets are currently congested (trapped). This value unlike the B^i_R and B^i_F has a weight decay term. This value gives the $B_T{}^i(t)$ a very short life span and congestion bias values will tend to zero as congestion clears. The bias adjustment for congestion is calculated as follows.

$$B_T{}^i(t) = B^i{}_T(t - 1) + \Delta B^i{}_T(t) - \varepsilon B^i{}_T(t - 1)$$

$$\Delta B^i{}_T(t) = \alpha_T \Sigma^k B_{block}$$

where

- α_T is a learning rate
- B_{block} is a bias vector representing blocked processors.
- ε is a weight decay term.

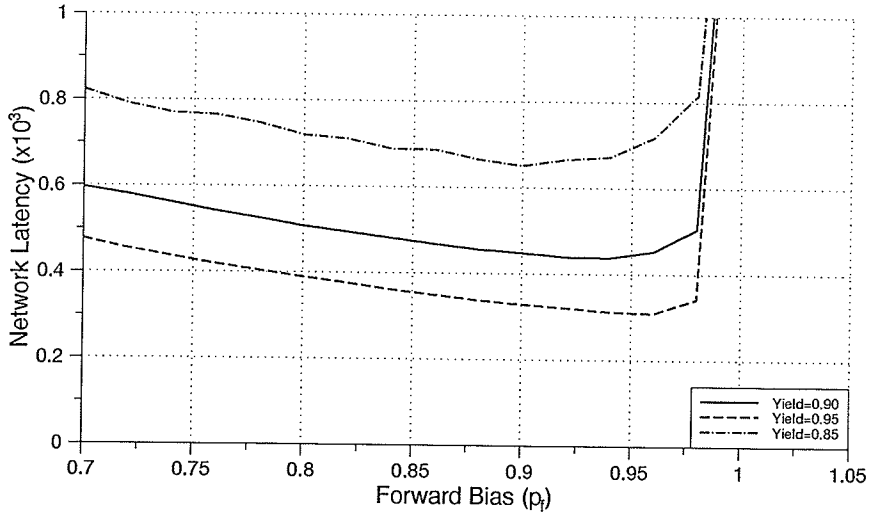
4.5 Simulation

4.5.1 Simulation Results

In order to demonstrate the effectiveness of adjusting the bias values as indicated in the previous section, we present the results of simulations of message-passing networks. Each of the three bias terms (B_R, B_F, *and* B_T) are adjusted independently to show its relation to network performance.

4.5.2 Constant Bias Simulation

In the first simulation a network of 625 (25 by 25 array) processors was simulated with 100 messages being delivered between randomly selected processors. The B_R value is adjusted such that forward bias term assumes values between 0.7 and 1.0. The results of these simulations are shown in Figure 22. For each of the three yield rates simulated, an optimal forward bias value (p_f) of 0.90, 0.94 and 0.96 is found for yield rates of 0.85, 0.90, and 0.95 respectively. For each of these cases a forward bias of 1.0 corresponding to shortest path routing with no backtracking left undelivered messages and hence infinite network latency. This graph shows that for different yield rates there is an optimum value of the forward bias.



B_R = (0.85,0.06,0.03,0.06)

a_0,a_1,a_2 = 0.8,0.1,0.1

$\alpha_F = 0.3$

$\alpha_T = 0.3$

$\varepsilon = 0.5$

Figure 22 Network Latency vs Bias

4.5.3 Anomalous Transport

Figure 23 shows the distribution of delivery times for packets in two similar networks, differing only in yield rates. In the case of the network with yield of 95% the distribution appears to be Gaussian as expected. For the case where the yield rate is 80% the distribution is clearly non Gaussian and is consistent with the transport times associated with anomalous transport as discussed in section 3.

A comparison of network latency versus yield is shown in Figure 18. Also shown is the conductivity of the network. As expected the network latency is inversely proportional to the conductivity. Although packets are still able to reach their destination, the expected delivery time of the packet is significantly degraded as yield decreases.

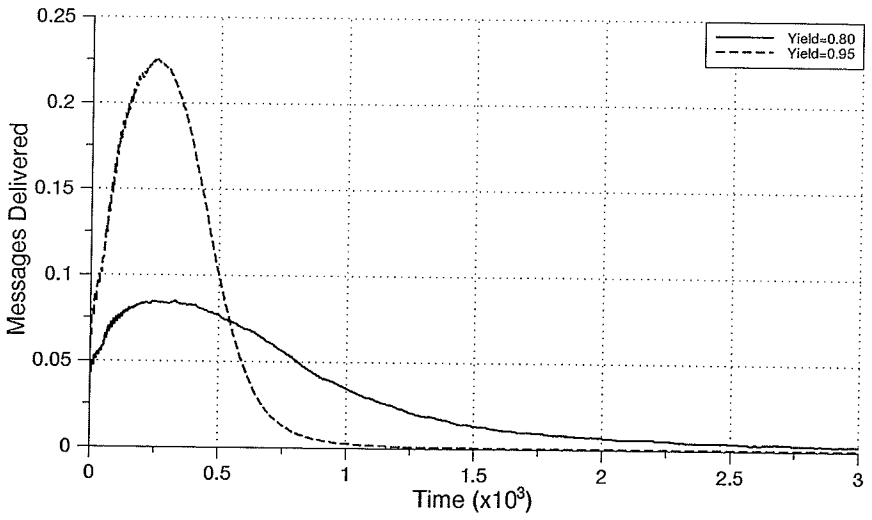


Figure 23 Anomalous Transport

4.5.4 Adaptive Bias

The following shows the relative performance improvement of the adaptive bias technique applied to the two cases shown in Figure 20(a) and (b). A 25 by 25 processor array was used for both cases.

This results are based on statistics generated from 1000 simulations for both algorithms.

The following values were used in the simulations:

	Constant Bias	Modified Bias
Case(a)	405	351
Case(b)	487	380

Table 2: Simulation Times

4.6 Virtual Cut-Through Routing

The nondeterministic adaptive routing algorithm can be easily extended to work as a virtual cut through routing algorithm.

Unfortunately, this non deterministic routing technique is not extendable to wormhole routing algorithms. As we will see in Chapter 5, wormhole routing is very susceptible to deadlock when cyclic dependencies exist in the channels of a network. To avoid deadlock, messages must not allocate channels which form a cycle. The very nature of the nondeterminstic routing algorithm which allows all channels to be used at any node by any message, will certainly allow cycles to exist. In fact as a simple example, a message may bounce back and forth between two adjacent processors a couple of times. This is clearly a cyclic condition, and will create deadlock in wormhole routing.

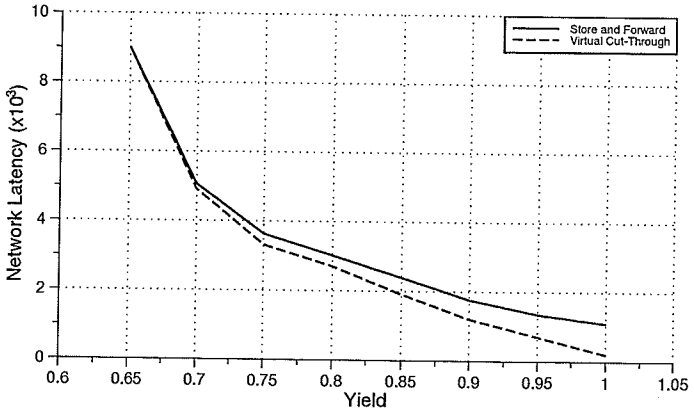


Figure 24 Store and Forward vs Virtual Cut-Through

Figure 24 shows a simulation of network latency versus yield for both the nondeterministic adaptive routing algorithm in the store and forward mode and in the virtual cut-through mode. For yield values close to 100%, we find that the virtual cut-through method is clearly superior to the store and forward method. However, the disparity between the two methods is less significant as the yield rate drops. This is to be expected, as the more that a message is blocked, the closer the network latency of the two methods will be to each other. As messages encounter faulty elements, and especially as messages traverse through bridges and articulation points in the network, there will be congestion in these spots. When a message is blocked, it is buffered as in the store and forward routing algorithm and the performance should be comparable.

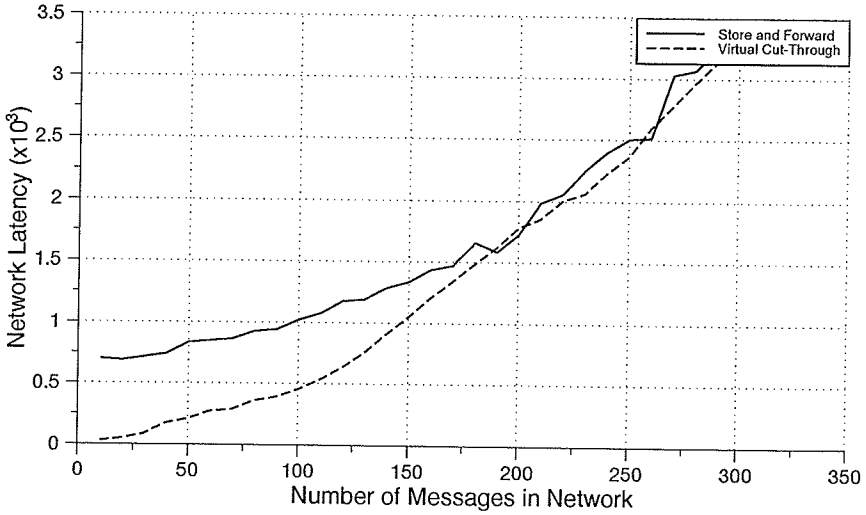


Figure 25 Performance Comparison Based on Load

A similar type of comparison is shown in Figure 25. In this simulation the network latency is shown as a function of the number of messages in the network. In this simulation, a constant yield rate of 95% is maintained, and the network latency for various amounts of traffic are shown for the two routing algorithms. Similar to the results in Figure 24, we see that the network latency is superior in the virtual cut-through algorithm for low traffic simulations, but the disparity disappears in higher traffic cases.

4.7 Summary

In this chapter we examined the use of percolation theory in the analysis of processor array networks. It found that percolation theory can be used to place bounds on the performance characteristics of these networks. Specifically we found that for two dimensional mesh architectures, we must have a yield rate higher than 60% (the percolation threshold) in order to maintain connectivity between functional processors. In addition, the conductivity of the network was found to be a useful tool in judging the ability of a network to permit message flow.

Also in this chapter we explored the use of a nondeterministic routing algorithm to route in the highly defective environment. The algorithm closely related to the biased random walker can

successfully route messages between source and destination processors (assuming they are connected), however the network latency was found to be unacceptable in low yield rate cases. Modifications were introduced to the algorithm to attempt to improve performance, by making the algorithm quasi-local, and utilizing algorithms to adjust the routing biases based on network congestions and faults.

Although we found that it is possible to implement routing algorithms for the highly disordered environment of WSI, if we wish to implement our goal of a general purpose parallel processing environment on a wafer, we will need to improve communications between processors. The bounds placed us by percolation theory show that we will have to consider alternative approaches. In later sections we will look at applying reconfiguration techniques in order to improve the characteristics of the network.

CHAPTER 5: Reconfigured Networks[1]

In the previous chapter we discussed routing in a disordered network where both faulty processing elements and interconnects exist. Although we introduced an algorithm which was able to route messages between connected processing elements in the network, the performance of the networks was not sufficient for most applications. In this section we will examine networks in which reconfiguration has been employed to increase the conductivity of the network. We utilize a reconfiguration scheme which gives us nearly 100 percent harvest of functional processors, but does not preserve the mesh topology. In this section we will look at the case where only processors are faulty. Interconnect failures will be handled in subsequent chapters.

Also, in the previous chapter we considered only store-and-forward and virtual cut-through routing algorithms. The highly disordered nature of the faulty array necessitated the use of nondeterministic routing algorithms which were able to successfully route between all processors which were in the spanning cluster of the network. In this section we will attempt to develop wormhole routing algorithms for faulty arrays.

5.1 Wormhole Routing

Wormhole routing attempts to improve network performance by eliminating the buffering of complete messages at each node as they traverse the network. The nondeterministic routing technique of the previous chapter is not easily adapted to wormhole routing techniques due to problems of deadlock which can occur when messages form a cycle. In the store-and-forward and virtual cut-through techniques, deadlock only occurs when all buffers in the network are full; a message could always be routed to any adjacent processing element if its buffer was not at capacity. A different scenario arises in wormhole routing where each message allocates a set of channels while it moves through the network. Once a channel is allocated to a message, it can only route flits from that message until the entire message has progressed through the channel. It is this allocation of multiple channels which gives rise to deadlock conditions.

With store-and-forward routing techniques we were able to assume that a network was constructed by using either a single bi-directional connection or two unidirectional connections between adjacent processing elements. With wormhole routing, we are restricted to networks in which only unidirectional connections are used.

It has been shown that the Channel Dependency Graph (CDG)[19] can be used to determine deadlock configurations in wormhole routing algorithms. The CDG is a directed graph where each node represents one channel in the graph. A directed edge exists between two nodes, if a mes-

1. Material from this chapter and chapter 6 has been accepted at the 1995 IEEE International Conference on WSI.

sage can be routed from the channel represented by the first node into the channel represented by the second node. If the CDG is acyclic, then the routing algorithm is guaranteed to be deadlock free. If a cycle (a path following the edges of the graph and both starting and terminating at the any node) exists then the routing algorithm is subject to deadlock. Figure 26(a) shows the eight unidirectional connections associated with each processing element. Assuming that each input channel (b,d,f,h) can route a message to each output channel (a,c,e,g), the CDG for a single node is illustrated in Figure 26(b). It can easily be seen that if each processing element implements all possible routing relations (input - output combinations) then cycles will exist between neighbouring processing elements.

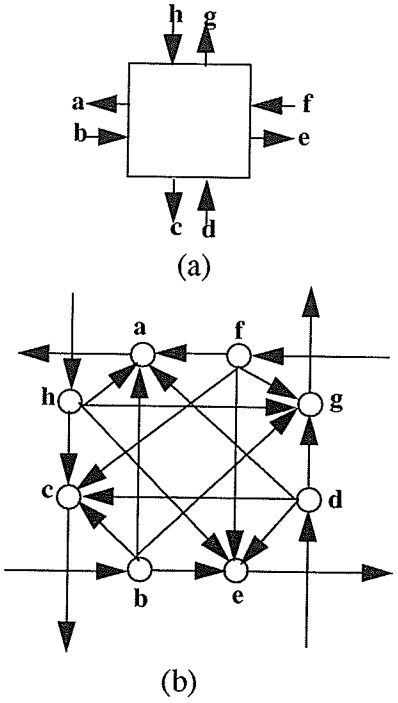


Figure 26 Channel dependency of a Processing Element

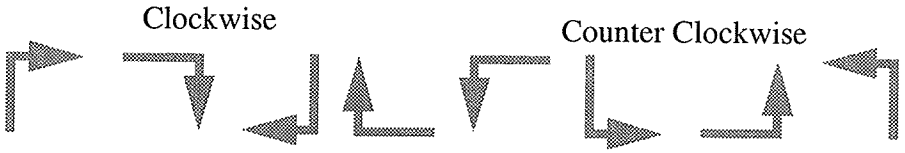
The CDG of a network with respect to a routing algorithm can tell us if a deadlock condition can exist in the network, although a cyclic CDG does not guarantee that a routing algorithm will produce a deadlock configuration. In [19] it was shown that any oblivious routing algorithm will be subject to deadlock if there exists a cycle in its CDG. This condition does not hold true for adaptive routing algorithms[21][31]. In this chapter we will only be concerned with oblivious routing algorithms, and hence we will be subject to deadlock whenever a cycle exists in the CDG.

5.2 Turn Model

In this section we will briefly give an overview of the turn model [29] which can be used in the analysis of wormhole routing algorithms. The model allows an easy method to determine if

any cycles exists in the channel dependency graph (CDG), and hence whether the algorithm is deadlock free or not. The basic concept in the Turn Model is to consider only the turns that a message can make while traversing a network. A turn refers to when a message flows from one channel (connection) to another channel which is oriented in a different direction. For example if a message enters a processing element from a channel connected to the bottom, and exits on a channel on the right, the message is said to have made a turn. There are eight ninety degree possible turns which can be modelled. In addition, there are also four zero degree and four one hundred and eighty degree turns which are possible in a two dimensional mesh. In Figure 27, the eight possible ninety degree turns are shown, and a set of labels (both numeric and character) are shown. It should be noticed that each of the 16 possible turns (of all degrees) corresponds to one of the arcs in the CDG model for a processing element shown in Figure 26.

The numeric labels are assigned such that a single bit reflects each of the four turns in an allowed direction. Turns are classified into two directions: Clockwise and Counter Clockwise. Each combination of three turns taken from the same direction will form an equivalent turn in the opposite direction. The numeric labels were chosen so that the equivalence operation can be performed by an logical complement of numeric label. The character labels for each turn consists of two letters, the first letter representing the original direction, chosen from the set (U: Up, D: Down, R: Right, and L: left). The second letter represents the new direction after the turn.



8 Possible 90 Degree Turns

UR	RD	DL	LU	LD	DR	RU	UL

Turn Labels

1000	0100	0010	0001	0010	0100	1000	0001

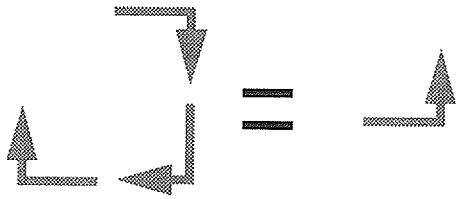
Numberic Labels

Figure 27 Turn Model

The main power of the Turn model is its ability to easily put a set of constraints on allowed turns that a routing algorithm may take, while remaining deadlock free. If we consider all the turns that a routing algorithm makes, if no cycles are formed (either clockwise or counter clockwise), then the routing algorithm is deadlock free. The converse if not true; however, it is possible to have a heterogeneous routing algorithm which utilizes all possible turns. The allowed turns must be placed so that no cycles form in the CDG.

In order to ensure no cycles are present in the network, we must not only be concerned with cycles which form in each of the two directions, but cycles which can form from combining turns

from each direction. Each three turns in one direction will have an equivalent turn in the opposite direction. One such relation is shown in Figure 28.



0111 = 1000

Figure 28 Turn Equivalence

5.3 Wormhole Routing without Reconfiguration

In this chapter we will be attempting to develop a wormhole routing algorithm which can be used in a reconfigured network. In the previous chapter we had developed a store and forward routing algorithm for networks with both faulty processing and interconnect elements but we wish to decrease the network latency. The non deterministic algorithm of the previous chapter will of course work in a reconfigured network, but we wish to restrict ourselves to algorithms which will route using close to minimal paths.

Before discussing the reconfiguration of processor arrays, we wish to show that it would be impossible to implement a wormhole routing algorithm in a unreconfigured array. This is important as it will give some insight as to what characteristics of a reconfigured network must exist for wormhole routing to be implemented.

Theorem 1

Any fault free mesh (m,n > 1) with a homogeneous routing function utilizing more than six distinct ninety degree turns will contain a cycle in its CDG.

Proof of Theorem 1

We will show that any homogeneous routing function utilizing seven ninety degree turns will contain at least one cycle. Since we have 7 turns, and there are only four distinct turns in each direction, we must have four turns in one direction, and three turns in the other direction.

This theorem can now be proved by simply considering any four connected processors as shown in Figure 29(a). We know that any four distinct cycles in the same direction will make a cycles. Since there are only two possible directions (clockwise and counter clock-

wise), we know that if seven distinct ninety degree turns are utilized, then either there are

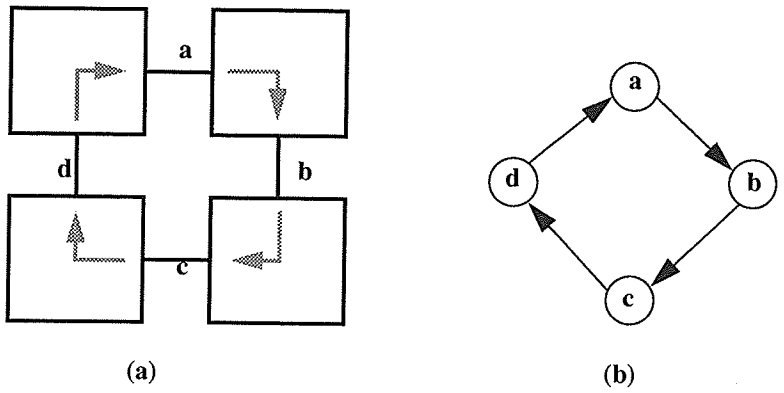


Figure 29 Cycle in Mesh

four distinct 90 degree turns in the clockwise direction and three in the counter clockwise direction, or there are three distinct 90 degree turns in the clockwise direction and four in the counter clockwise direction. In either case there are at least four distinct turns in one direction, and hence there must be a cycle in either the clockwise or counter clockwise direction. In the figure we show the case where there is a clockwise cycle. Since there are always four connected processing elements as shown in a mesh with m,n > 1, we know there is always a cycle. The CDG for the connected processing elements is shown in Figure 29(b).

■

Although a fault free mesh must always contain a cycle if it utilizes more than six ninety degree turns, the converse is not true. It is possible for the presence of faults to remove the cycles. As an example consider the small network shown in Figure 29. In this example, if all processing elements are allowed to use all eight ninety degree turns, the presence of faults and the borders of the network limit the actual turns which can be used. Cases like these are the exception. In this case, the faults and borders of the network have transformed the routing algorithm from a homogeneous one to a heterogeneous one.

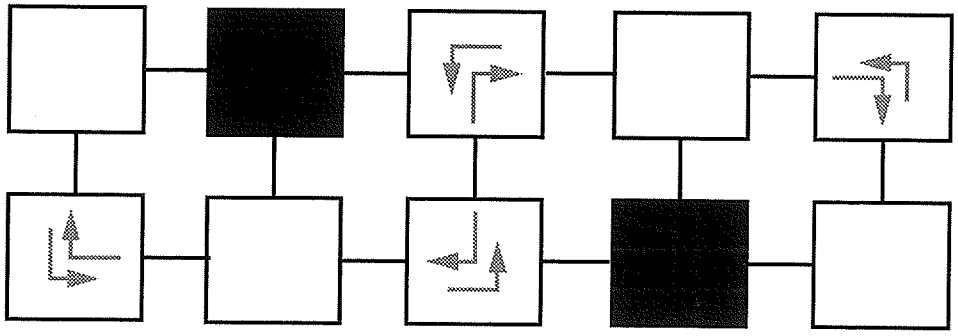


Figure 30 8 turns no cycle

Theorem 2

No local homogeneous deadlock free wormhole routing algorithm is complete and fault tolerant for a single processor fault in a 2-D mesh network..

Proof of Theorem 2

We know from Theorem1 that at most 6 turns can be allowed in any deadlock free homogeneous routing algorithm. With more than 6 turns, there must be at least one cycle.

If we show that no local homogeneous algorithm using 6 turns can route deadlock free in a mesh with a single fault, then no algorithms using less than 6 turns can route deadlock free, since these will be subsets of the 6 turn algorithm.

Let A be a four bit label for a clockwise turn, and let B be a 4 bit label for a counter clockwise turn. We use the labelling shown in Figure 27. A will represent the turn in the clockwise cycle which is not allowed by the routing algorithm, and likewise B will represent the turn in the counter clockwise cycle which is not allowed by the routing algorithm.

It is important to note that $A \neq B$. The equivalent turn to the three turns in the clockwise cycle ($\overline{A}$) is A. Thus A must be allowed to turn in the counter clockwise cycle. Therefore $A \neq B$. We also know $A \neq 0000$, since this does not represent a valid turn.

Let R_{CW} represent the turns allowed in the clockwise cycle. $R_{CW} = \overline{A}$.

Let R_{CCW} represent the turns allowed in the counter clockwise cycle. $R_{CCW} = \overline{B}$.

Consider the network show in Figure 31. Four possible locations of a fault are shown. Each

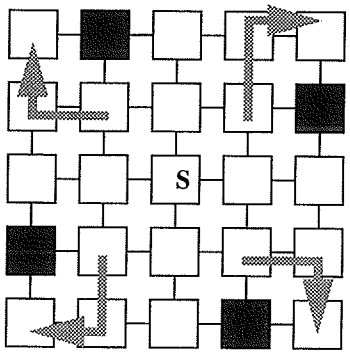


Figure 31 Example Network

of these faults represents one of the many places where a fault may occur (we are not implying that the four faults are in the network simultaneously, but showing four possible locations for a fault)). If we wish to route a message from the processing element marked S to any of the corner processing elements in the network, the four indicated turns must be present in the routing function. Each turn must be there since that turn represents the only way for a message to reach the corner processing element. Consider the case where the destination is in the upper right corner of the network, and the fault may be directly below it. In this case it must be possible to route to this processing element from the left adjacent processing element. This means that it must be possible for a message to be routed to the top row. A message can only be routed to the top row from below (there are no other connections). There are only two ways for a message to be routed from a lower row to the corner: A message can only be routed to the top row and then towards the corner by either making a UR turn or a combination UL and 180 degree turn. The later case is equivalent to a UR turn. This means that the turn 1000 (UR) in the clockwise direction must be allowed.

R_{CW} and $1000 = 1000$

similarly we can show that the turns RD, DL, and LU, all must be in the allowed turns.

R_{CW} and (UR or RD or DL or LU) = UR or RD or DL or LU

UR or RD or DL or LU = 1111

$\overline{R}_{CW} = 0000$

$A = 0$

Since A can not be equal to 0, no turn in the clockwise direction can be excluded from the allowed turn list. Similarly no counter clockwise turn can be excluded.

Since the algorithm is homogeneous, we know that the presence of the four turns, which form a clockwise cycle, indicates the algorithm is subject to deadlock. A similar argument may be made for the counter clockwise direction.

■.

Theorem 3

A non-local deadlock-free routing algorithm exists for all processors in a strongly connected component of the network

Proof of Theorem 3

We will prove this theorem by showing a technique to find a non-local heterogeneous routing algorithm which will route between all processing elements in a strongly connected component. Our routing algorithm will be based on a tree obtained by a depth first search of the strongly connected component.

We will use the following algorithm to calculate the routing table. This algorithm is a slightly modified version of the Depth First Search (DFS)[18]. This algorithm will produce a two dimensional table route_table which will hold the channel to be routed. The indexes correspond to the source and destination processing element respectively.

```
DFS_SCC(G)
    FOR EACH P_i ∈ P DO
        status[P_i] = untouched
        FOR EACH P_j ∈ P DO
            route_table[P_i][P_j] = NIL
    DFS_VISIT(P)
DFS_VISIT(P_A)

    if (status[P_A] = found) return;
    status[P_A] = found
    FOR EACH P_B ∈ A^1[P_A] DO
        IF status[P_B] = untouched
            FOR EACH P_i ∈ P, status[P_i] = found DO
                route_table[P_B][P_i] = channel(P_B,P_A)

            FOR EACH P_i ∈ P, status[P_i] = untouched DO
                route_table[P_B][P_i] = channel(P_A,P_B)

            DFS_VISIT(P_B)
```

We will now show that this algorithm will produce a routing algorithm without cycles. The DFS algorithm will produce a spanning tree of the network. We have shown an example tree in Figure 32, and a linearized version of the tree. Notice that the non leaf nodes are duplicated in Figure 32(c). The nodes of this graph represents processors, and the edges channels. We can create a CDG from this graph. If we do not allow leaf nodes to be intermediate points in the paths of messages, the CDG will be acyclic, and therefore the routing algorithm will be deadlock free.

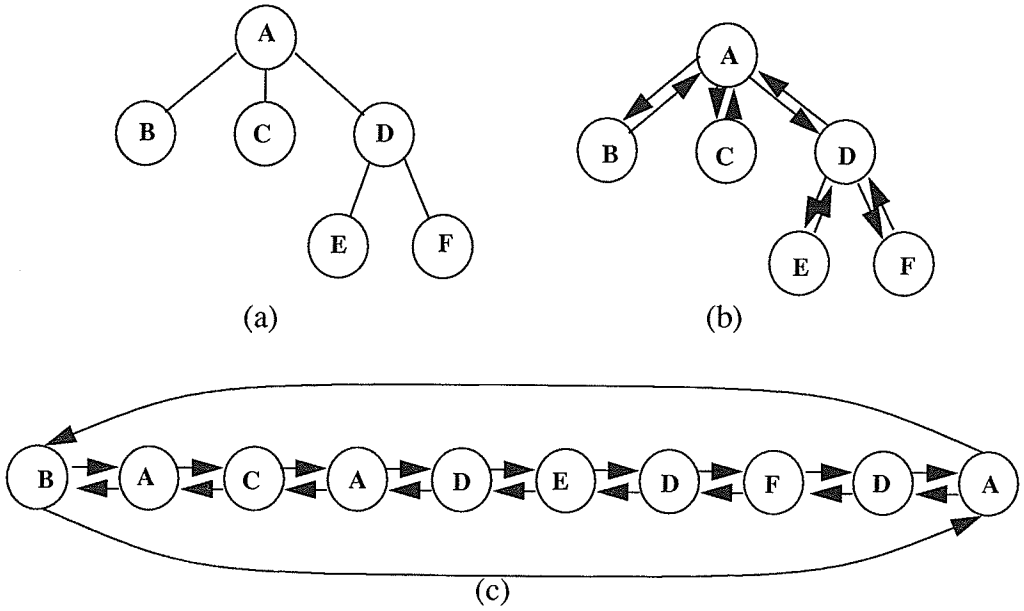


Figure 32 Linearized Tree

■

Although this theorem gives us a method to implement a deadlock free routing algorithm this is not an efficient way to implement a routing algorithm as it only uses a small subset of available connections for routing. The subgraph used by the routing algorithm is a tree and as such has a bisection width of 1. This theorem tells us that it is always possible to implement a deadlock free routing algorithm on a network which is strongly connected. If a network is not strongly connected, there exists at least one processing element which is not reachable from other processing elements, and hence no algorithm will exist. If we wish to consider only local routing algorithms, it is not always possible to find such a routing algorithm (we will prove this later in this chapter).

Local heterogeneous deadlock free wormhole routing algorithms exist which will route all permutations in a 2-D mesh network with a single faulty processor. We will not prove this, but will offer an example algorithm later in this chapter which demonstrates this. It is important to note is that Heterogeneous routing algorithms are more powerful than homogeneous routing algorithms. This can be easily verified, as homogenous routing algorithms are a subset of heterogeneous routing algorithms.

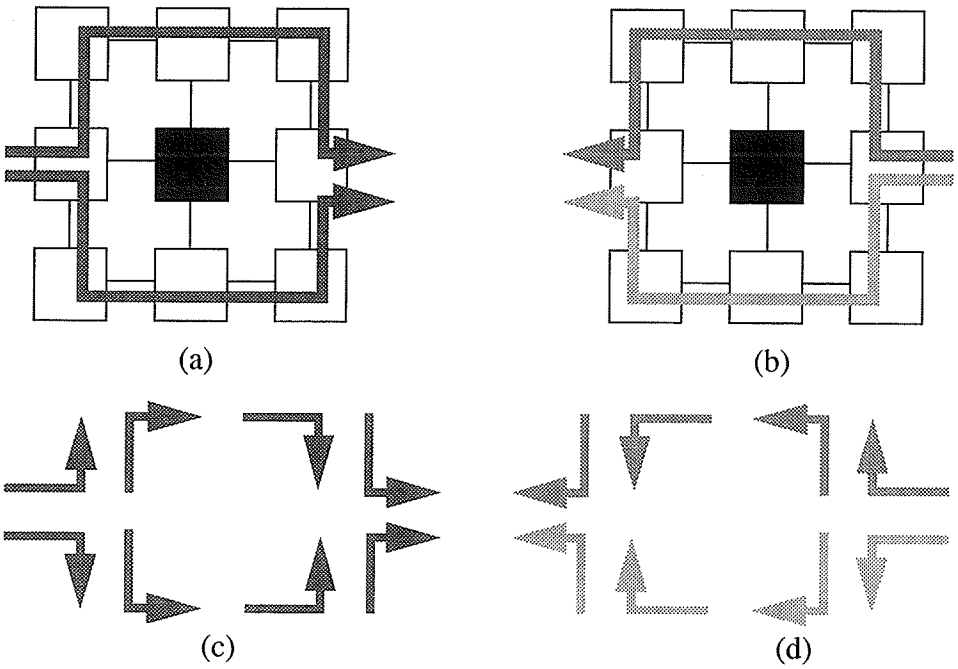


Figure 33 Routing around Faults

The difficulty of routing around a single fault is illustrated in Figure 33. In Figure 33(a) and (b) we show the paths a message travelling along the horizontal axis might take to bypass a fault. For each direction the message is travelling there are two paths which the message can use to bypass the fault. In Figure 33(c) and (d) we show the turns associated with the bypass paths. It can be seen the both paths use the same four turns. It can also be seen that the four turns associated with (a) in addition to the four turns associated with (b) will utilize all eight turns, and thus create a deadlocked routing algorithm.

In spite of this difficulty, routing algorithms have been developed which will route in the presence of a single fault[30]. This is accomplished by avoiding the situation shown in Figure 33, where a message travels along the axis to its destination. Instead a message will attempt to route towards its destination, but will avoid the axis of the destination as long as possible. These types of algorithms will works for single faults, but offer no advantage for multiple faults. Since we are concerned with routing in the presence of multiple faults, we will not use this technique but investigate alternative techniques.

5.4 2-D Diogenes Reconfiguration

Diogenes reconfiguration[71] was originally proposed as a reconfiguration methodology to reconfigure faulty 2 dimensional arrays into a fault free 1 dimensional linear array. Reconfiguration is accomplished by utilizing bypass interconnects and switches which may be set to bypass

and isolate faulty elements. Figure 34 shows the reconfigured (horizontally) one dimensional array, and the corresponding logical linear array. Figure 35 shows the one dimensional Bypass interconnect.

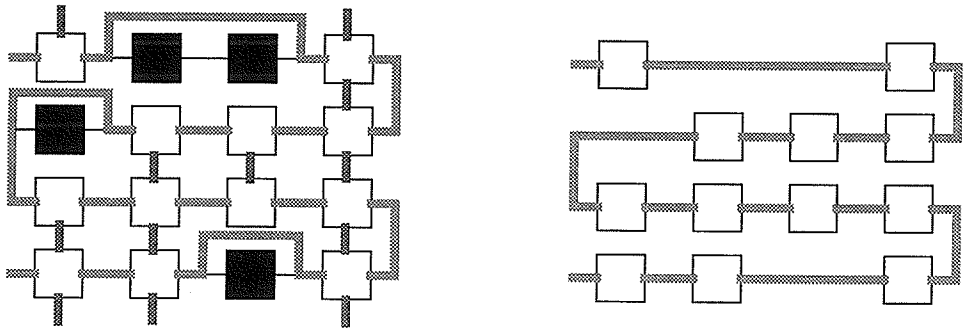


Figure 34 One Dimensional Diogenes

The advantages of the Diogenes approach include:

1. All functional processors are used in the reconfigured network.

2. The bypass interconnect scheme is simple and requires minimal overhead.

Diogenes reconfiguration suffers three important limitations:

1. Faults can not occur in the interconnect. Any fault in the interconnect (either in the normal or the bypass connections can potentially prevent successful reconfiguration.
2. Produces a linear array out of a two dimensional array. This techniques is only useful if a one dimensional structure is required.
3. The delay between adjacent operational processors in the reconfigured array is unbounded.

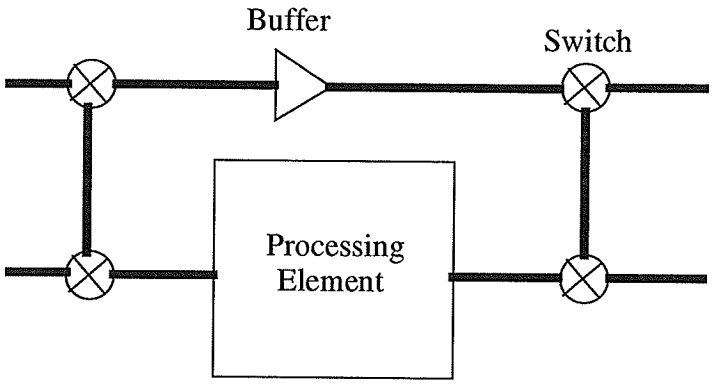


Figure 35 Positive X direction Bypass Connection

As we are interested in asynchronous systems, the unbounded delay between adjacent processing elements in the reconfigured array will not stop operation of the network, however, it may slow down operation of the channel to the point where it is unrealistic to use the channel. This will cause the channel to either be busy for extended periods of time (and cause congestion) or be so slow that the channel can be considered faulty. In either case we will be looking at adaptive routing algorithms which can avoid congested channels, and will look at reconfiguration of faulty channels in the next chapter. Since we are interested in two dimensional networks, and wish to incorporate fault tolerance in interconnects, the simple 1 dimensional Diogenes approach will not be sufficient.

Diogenes can be extended to two dimensional arrays by performing one dimensional reconfiguration in both the horizontal and vertical directions. This is shown in Figure 36.

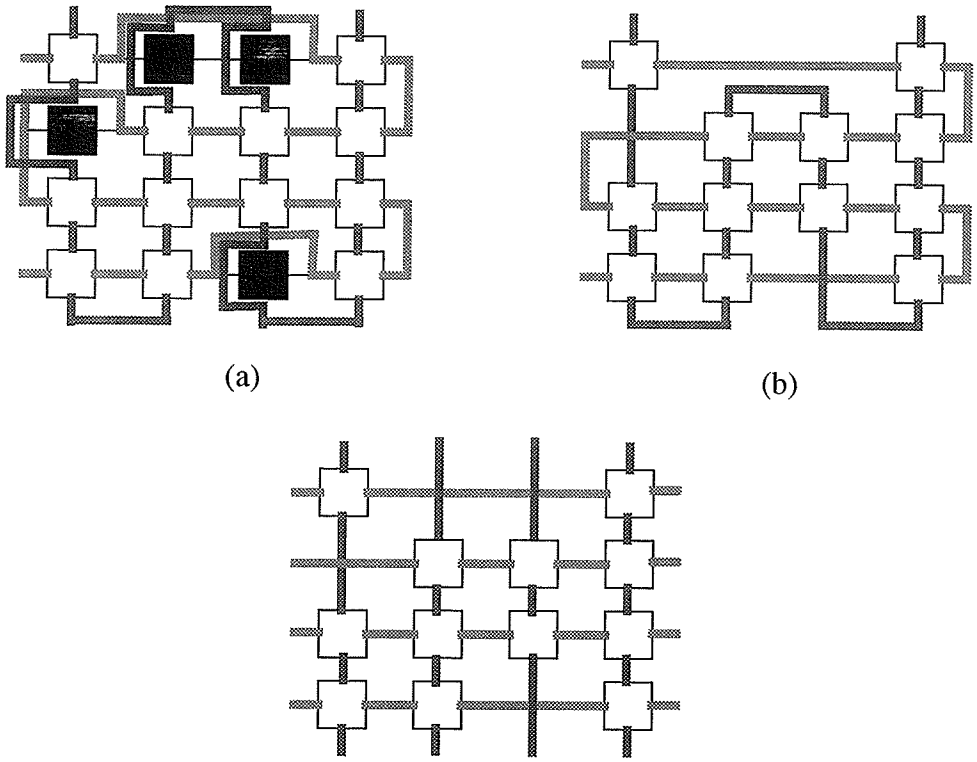


Figure 36: Two dimensional Diogenes

In the two dimensional reconfiguration, it is not necessary to connect alternating rows and columns, although the extra connectivity can be exploited. The two dimensional reconfiguration has the following properties:

1. All functional processing elements are utilized.
2. The reconfigured topology is irregular.
3. In the presence of faulty connections, one or more processing elements may be isolated from the rest of the network.

If a connection is faulty, we simply remove it from the network. Without faults in the connections, we could always route message between any pair of processors by following the linear array (either horizontally or vertically) between the processors. With faults in the connections between processors, our network becomes much more disordered, and maybe even disjoint.

We would normally assume that we have a unit delay between adjacent processing elements, and we measure the distance between adjacent processing elements as one. In the case of a reconfigured network, the distance and the delay between processors is not so clear. We have previously defined three distance measures: d_{min}, d_r, and d_R. In the case of a reconfigured network, the d_{min} measure is reduced.

We need to define a new representation for a path. Whereas the definition for Q_p was the sequence of processors through which a message transversed, we will also find it convenient to have a definition of a path which also includes the faulty processing elements which are bypassed by the bypass connections after reconfiguration.

A path between two processors P_S and P_D is represented by Q_{p*}.

> Let Q_{P*} represent a path between two processing elements. We represent Q by the sequence of processing elements through which the path traverses and the faulty processing elements which are bypassed by bypass connections.
>
> $Q_{P*} = P_1, P_2, P_3, ..., P_n$

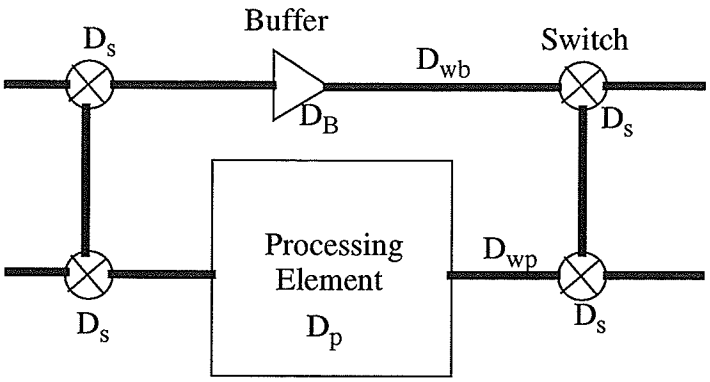


Figure 37 Delays in Connections

The delay in a reconfigured network is more difficult to model at an abstract level. Without knowing exact implementation details about the network an accurate model of delay is not possible. In this thesis we will assume that the delay of a path is proportional to the number of operational processing elements in the path, delay(Q) $\propto |Q_p|-1$, in a similar manner to the delay associ-

ated with the delay in a non reconfigured path. This requires that the delay of each bypass interconnect be negligible compared to the delay associated with each processing element. We justify this assumption by assuming we are using a buffer in the bypass interconnect and modelling the delay as follows:

$$delay(Q) \propto |P_\alpha - 1|(D_p + D_{wp} + D_s) + |P_\beta|(D_{wb} + D_B + 4D_s)$$

where

D_{wp} and D_{wb} are the wire delays associated with interprocessor and bypass connections respectively. D_p is the delay associated with the switching and buffering circuits in the processing element. D_s is the delay associated with the switches in the interconnect. D_B is the delay associated with the buffer in the bypass connection.

$P_\alpha = \{p_i | p_i \in Q_{P*}$ and $p_i \in P_O\}$. This is the set of functional processors along the path Q.

$P_\beta = \{p_i | p_i \in Q_{P*}$ and $p_i \in P_f\}$. This is the set of faulty processors along the path Q.

We can assume that the delay from a functional processing element is much greater than the delay from the simple switch and the small interconnect wire between adjacent processing elements, since the buffering and switching circuits will be fairly complex. For a bypass connection, it is safe to assume that the wire delay will be the significant factor, compared to the small delay from the buffer and switches. Thus:

$D_p >> D_{wp}$ and $D_p >> D_s$

$D_{wb} >> D_B$ and $D_{wb} >> D_S$

$delay(Q) \approx |P_\alpha - 1|D_p + |P_\beta|D_{wb}$

$delay(Q) \propto |Q_{p*}| - 1$, assuming $D_p \approx D_{wb}$

$delay(Q) \propto |Q_p| - 1$, assuming $D_p >> D_{wb}$

If we compare the delay of the processing element with the delay of a bypass connection, we will notice that in both cases, an almost equal amount of wire is required (both have to route information from one side of the processing element to the opposite), and thus any delay associated with wire length will affect both routes equally. The processing element will have to have control circuitry and buffers along on the internal path, thus it is safe to assume that $D_p > D_{wb}$. If wire delays are the prominent source of delay, then clearly $D_p \approx D_{wb}$, otherwise $D_p >> D_{wb}$.

This model of the delay gives us a delay associated with a reconfigured connection proportional to the number of bypassed faulty processing elements. Although delay is unbounded (our current fault assumptions allow an unlimited number of consecutive faulty processing elements) the delay will not affect operation or performance of the network, since all interprocessor communication is asynchronous.

Theorem 4

Consider two processing elements p_s and p_d such that they are both in the same column. Then $d_{min}(p_s,p_d) \leq d_r(p_s,p_d)$.

Proof of Theorem 4

We consider the shortest path Q_p from $p_s(x_s,y)$ to $p_d(x_d,y)$. This is the path which follows the x axis from source to destination.

We will assume without loss of generality that $x_s \leq x_d$.

We define two sets of processors P_α and P_β which are the processing elements located in between the source and destination processing elements including and not including faulty processing elements respectively.

Let $P_\alpha = \{p_i(x_i,y) | \ x_s \leq x_i \leq x_d\}$

Let $P_\beta = \{p_i(x_i,y) | \ p_i \in Q_P, x_s \leq x_i \leq x_d\}$

We know that all processing elements in P_β are in P_α, $P_\beta \subseteq P_\alpha$

We also know that any elements in P_α and P_F are not in P_β

$(P_\alpha \cap P_F) \cap P_\beta = \varnothing$

Therefore

$|P_\beta| + |P_\alpha \cap P_F| = |P_\alpha|$

$|P_\beta| \leq |P_\alpha|$

$d_{min}(p_S,p_d) \leq d_r(p_s,p_d)$

■

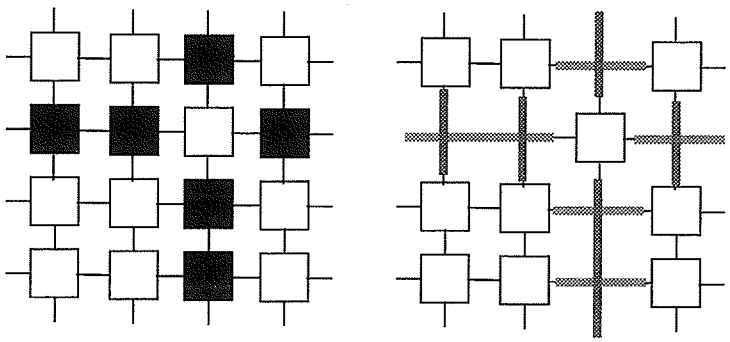
The importance of this theorem is that it shows that it is not possible to use local information to find the optimal (shortest) path between source and destination. Since the only information available at any processor is the location of the current processing element, and the location of the destination, and since the shortest path may be independent of this information, it is not always possible to find the shortest path.

Corollary 2.1

No greedy local routing algorithm will be optimal

Although the Diogenes reconfiguration will allow us to use all functional processors in the network, it is still possible that the network will contain processor subsets which are not able to communicate with each other. Each of these subsets will be strongly connected As an example,

consider the network shown in Figure 38. A processing element exists which is isolated from all



(a) (b)

Figure 38 : Isolated Processing Element

other processing elements in the network, although this processing element is still useable, as it has connections to external connections. Had we utilized connections on the edge of the network to maintain a linear array in both the horizontal and vertical directions, this processing element would have been connected to the other processing elements, and we would have obtained 100 per cent harvest of processors.

Theorem 5

As the network size increases, the probability that all functional processing elements are strongly connected will approach 1

Proof of Theorem 5

Consider two processors $P_A(x_A,y_A)$ and $P_B(x_B,y_B)$, such that $x_A \neq x_B$ and $y_A \neq y_B$.

Let p_c be the probability that P_A and P_B are strongly connected.

Let p_o be the probability that any processing element is operational. This is equivalent to the yield rate.

We know that P_A and P_B are connected if either $P_C(x_A,y_B)$ or $P_D(x_B,y_A)$ is operational, since all operational processing elements in a common row or column are strongly connected. The probability for this is $1- 2p_o + p_o^2$.

Likewise P_A and P_B are connected if any two processing elements $P_i(x_k,y_B)$ and $P_j(x_k,y_A)$, $0 \leq k \leq m$, $x_k \neq x_A$, $x_k \neq x_B$ are both operational. The probability for this is $1- (1-p_o^2)^{m-2}$.

Likewise P_A and P_B are connected if any two processing elements $P_i(x_A,y_k)$ and $P_j(x_B,y_k)$, $0 \leq k \leq n$, $y_k \neq y_A$, $y_k \neq y_B$ are both operational. The probability for this is $1- (1-p_o^2)^{n-2}$.

Thus $p_c \geq (1-p_o^2)^{m-2} + (1-p_o^2)^{n-2}$.

An inequality is used since this term does not represent all possible ways that P_A and P_B

can be connected (the term $1 - 2p_o + p_o^2$ has already be ignored).

The limit of p_c as either m or n approaches ∞ is 1.

■

We know that it is always possible to implement a deadlock free routing algorithm on any strongly connected component in a network. This theorem shows use that as the network size increases, all the processors will become strongly connected.

5.5 Wormhole Routing Algorithms

One of the major drawbacks of the nondeterministic routing algorithm discussed in Chapter 4 was its performance. Although able to route in an extremely hostile environment of faulty processing elements and interconnects with no reconfiguration, packets travelling large distances may spend most of there time traversing paths which may not lead to the intended destination. With the diogenes reconfigured network discussed in the previous section, much more order exists in the network, and thus we should be able route more efficiently from source to destination. In this section we will introduce a wormhole routing algorithm for the reconfigured network. Wormhole routing was not possible in the non-reconfigured mesh.

In order to implement a wormhole routing algorithm for the Diogenes reconfigured network, we most develop an algorithm which avoids deadlock.

One important property of a 2-D Diogenes reconfigured network is that any routing algorithm developed for unreconfigured networks can be used on a reconfigured network.

Theorem 6

Any path Q, which exists between two processing elements $P_S(x_S,y_S)$ and $P_D(x_D,y_D)$ in a faulty mesh will also exist in the reconfigured mesh.

Proof of Theorem 6

Consider any path Q_P generated by a routing algorithm.

$Q_P = P_1,P_2,P_3,...,P_n$ such that $P_1 = P_S$ and $P_n = P_D$

Since the path Q_P will route in a faulty mesh, we know that $P_1,P_2,P_3,...,P_n \in P_O$.

We also know that since $d(P_i,P_{i+1}) = 1$, $0 \leq i < n-1$, that these two processors will also be connected by a normal connection.

Thus we know that any path which exists in the faulty mesh, will also exist in the reconfigured mesh.

■

Another interesting aspect of implementing routing algorithms in the reconfigured network

is that we can implement a different strategy in routing. In fault free meshes, it is normally desirable not to route along the axis as in XY routing. As we saw earlier, this type of routing will have problems even in the presence of a single fault. However, in the reconfigured network, we know that all functional processors in a row or column can always route to each other. Thus the basic strategy of our routing will change. Before, the goal was to get to the destination avoiding the row and column of the destination; now, our goal will be to reach the row or column of the destination.

5.5.1 Homogeneous XY routing

It is fairly straightforward to implement a routing algorithm for the reconfigured network. In this section we will discuss a modified version of the XY routing algorithm[23][63] which has been adapted to route in the presence of faults in the reconfigured network.

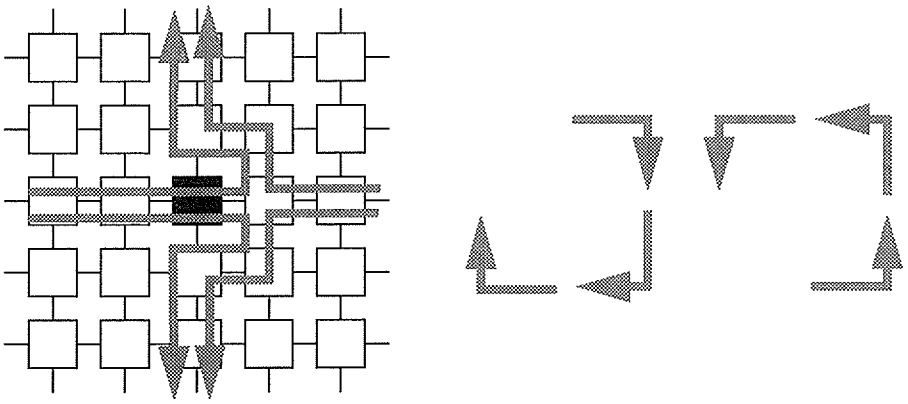


Figure 39 Diogenes Routing and Turns

We will now introduce the modified XY routing algorithm for the Diogenes reconfigured network. Recall that in the XY routing algorithm, a message is first routed along the x row until it reaches the y column which contains the destination processing element. The algorithm then routes the message along the y column to its destination. In the new algorithm, we have three phases of operation:

1. X Routing Phase. In this phase, a message is routed from the source processing element along the row until it reaches the column containing the destination processing element. If the processing element in this row which is also in the column of the destination processing element is faulty, we route to the first functional processor to the right of the faulty processor. Then we are in the column containing the destination processing element; the algorithm proceeds to the Y Routing Phase. Otherwise the algorithm proceeds to the Route to Y Phase.

2. Route to Y Phase. In this phase, we need to find a path to the column containing the desti-

nation. The algorithm simply proceeds along the current column one hop towards the destination processing element, and then the algorithms reverts back to the X Routing Phase.

3. Y Routing Phase. In this phase the message is now in the column containing the destination processor. The message can be routed along the column towards the destination.

In Figure 39, we show how the routing algorithm will route around a single fault, and the associated turns with the algorithm.

Theorem 7

The Modified XY routing algorithm is deadlock free

Proof of Theorem 7

We will use the turn model to model allowed turns a message can make, and we will use the results of [29] to show that these turns make a deadlock free routing algorithm. We utilize the following statements proved in[29]:

1. Six turns is the maximum allowed number of turns in deadlock free routing for singlely connected meshes. Not all sets of six turns are acyclic.

2. Three distinct turns in the same direction (clockwise or counter clockwise) can be modelled as one turn in the opposite direction.

3. The set of turns does not form a cycle if neither the clockwise or counter clockwise turns forms a cycle.

4. We must consider not only whether the turns make a cycle, but also whether the equivalent turns of combinations of allowed turns, with allowed turns make cycles in the CDG.

   Figure 39 (b) shows the six allowed turns that a message can make. As long as no cycles can be formed with these turns, we know that the CDG will be acyclic, and the algorithm deadlock free.

   Since the set of six turns meet all of the criteria above, the algorithm is deadlock free.

■

Theorem 8

The routing algorithm will successfully route in any network containing a single fault not on the right border (x=m-1) of the network.

Proof of Theorem 8

Let $p_s(x_s,y_s)$ be the source processing element, and let $p_d(x_d,y_d)$ be the destination. Furthermore let p_x be the processing element with a fault.

We know that p_x is not on the border of the network. Specifically

$p_x(x_i,y_j)$ has coordinates such that $i \neq m-1$

if $x_i \neq x_s$ and $y_d \neq y_j$

The algorithm will route the same as XY routing, and will not traverse the faulty processing element p_x.

$Q_p = p_s(x_s,y_s),p_l(x_{s+1},y_s),...,p_k(x_d,y_s),p_l(x_d,y_{s+1}),...,p_s(x_d,y_d)$ $x_s<x_d$ and $y_s< y_d$

$Q_p = p_s(x_s,y_s),p_l(x_{s-1},y_s),...,p_k(x_d,y_s),p_l(x_d,y_{s+1}),...,p_s(x_d,y_d)$ $x_s>x_d$ and $y_s< y_d$

$Q_p = p_s(x_s,y_s),p_l(x_{s+1},y_s),...,p_k(x_d,y_s),p_l(x_d,y_{s-1}),...,p_s(x_d,y_d)$ $x_s<x_d$ and $y_s> y_d$

$Q_p = p_s(x_s,y_s),p_l(x_{s-1},y_s),...,p_k(x_d,y_s),p_l(x_d,y_{s-1}),...,p_s(x_d,y_d)$ $x_s>x_d$ and $y_s> y_d$

$p_x \notin Q_p$ and for all $p_i \in Q_p$, $p_i \in G$

if $x_i = x_d$ and $y_s \neq y_j$

$Q_p = p_s(x_s,y_s),p_l(x_{s+1},y_s),...,p_k(x_d,y_s),p_l(x_d,y_{s+1}),..,p_q(x_d,y_{j-1}),p_r(x_d,y_{j-1}),..,p_s(x_d,y_d)$

$x_s<x_d$ and $y_s< y_d$

$Q_p = p_s(x_s,y_s),p_l(x_{s-1},y_s),...,p_k(x_d,y_s),p_l(x_d,y_{s+1}),...,p_q(x_d,y_{j-1}),p_r(x_d,y_{j+1}),..,p_s(x_d,y_d)$

$x_s>x_d$ and $y_s< y_d$

$Q_p = p_s(x_s,y_s),p_l(x_{s+1},y_s),...,p_k(x_d,y_s),p_l(x_d,y_{s-1}),...,p_q(x_d,y_{j+1}),p_r(x_d,y_{j-1}),..,p_s(x_d,y_d)$

$x_s<x_d$ and $y_s> y_d$

$Q_p = p_s(x_s,y_s),p_l(x_{s-1},y_s),...,p_k(x_d,y_s),p_l(x_d,y_{s-1}),...,p_q(x_d,y_{j+1}),p_r(x_d,y_{j-1}),..,p_s(x_d,y_d)$

$x_s>x_d$ and $y_s> y_d$

$p_x \notin Q_p$ and for all $p_i \in Q_p$, $p_i \in G$

if $x_i \neq x_d$ and $y_s = y_j$

$Q_p = p_s(x_s,y_s),p_l(x_{s+1},y_s),...,p_q(x_{i-1},y_s),p_r(x_{i+1},y_s),...,p_k(x_d,y_s),p_l(x_d,y_{s+1}),...,p_s(x_d,y_d)$

$x_s<x_d$ and $y_s< y_d$

$Q_p = p_s(x_s,y_s),p_l(x_{s-1},y_s),...,p_q(x_{i+1},y_s),p_r(x_{i-1},y_s),...,p_k(x_d,y_s),p_l(x_d,y_{s+1}),...,p_s(x_d,y_d)$

$x_s>x_d$ and $y_s< y_d$

$Q_p = p_s(x_s,y_s),p_l(x_{s+1},y_s),...,p_q(x_{i-1},y_s),p_r(x_{i+1},y_s),...,p_k(x_d,y_s),p_l(x_d,y_{s-1}),...,p_s(x_d,y_d)$

$x_s<x_d$ and $y_s> y_d$

$Q_p = p_s(x_s,y_s),p_l(x_{s-1},y_s),...,p_q(x_{i+1},y_s),p_r(x_{i-1},y_s),...,p_k(x_d,y_s),p_l(x_d,y_{s-1}),...,p_s(x_d,y_d)$

$x_s>x_d$ and $y_s> y_d$

$p_x \notin Q_p$ and for all $p_i \in Q_p$, $p_i \in G$

if $x_i = x_d$ and $y_s = y_j$

$Q_p = p_s(x_s,y_s),\ p_l(x_{s+1},y_s),...,p_k(x_{d-1},y_s),\ p_k(x_{d+1},y_s),\ p_l(x_{d+1},y_{s+1}),\ p_l(x_{d+1},y_{s+1}),...,$
$p_s(x_d,y_d)$ $\quad x_s<x_d$ and $y_s< y_d$

$Q_p = p_s(x_s,y_s),p_l(x_{s-1},y_s),...,p_k(x_{d+1},y_s),p_k(x_{d+1},y_{s+1})p_l(x_d,y_{s+1}),...,p_s(x_d,y_d)$

$\quad x_s>x_d$ and $y_s< y_d$

$Q_p = p_s(x_s,y_s),\ p_l(x_{s+1},y_s),...,p_k(x_{d-1},y_s),\ p_k(x_{d+1},y_s),\ p_l(x_{d+1},y_{s-1}),\ p_l(x_{d+1},y_{s-1}),...,$
$p_s(x_d,y_d)$

$\quad x_s<x_d$ and $y_s> y_d$

$Q_p = p_s(x_s,y_s),p_l(x_{s-1},y_s),...,p_k(x_{d-1},y_s),p_l(x_{d-1},y_{s-1}),p_l(x_d,y_{s-1}),...,p_s(x_d,y_d)$

$\quad x_s>x_d$ and $y_s> y_d$

$p_x \notin Q_p$ and for all $p_i \in Q_p$, $p_i \in G$

Since this covers all possible cases, the algorithm will successful route in the presence of a single fault.

■

We placed a restriction on the algorithm that the fault can not be on the right edge of network. An example of why this restriction is placed on the network can be seen in the proof of the previous theorem. Consider the case where $x_i = x_s$, $y_d = y_j$, $x_s<x_d$, and $y_s< y_d$. The path Q_p will be $Q_p = p_s(x_s,y_s),\ p_l(x_{s+1},y_s),...,p_k(x_{d-1},y_s),\ p_k(x_{d+1},y_s),\ p_l(x_{d+1},y_{s+1}),\ p_l(x_{d+1},y_{s+1}),...,\ p_s(x_d,y_d)$. If the fault is located on the right edge, the coordinate will be $p_x(x_m,y_s)$ and $x_m = x_d$, and thus the path can be written as: $Q_p = p_s(x_s,y_s),\ p_l(x_{s+1},y_s),...,\ p_k(x_{m-1},y_s),\ p_k(x_{m+1},y_s),\ p_l(x_{m+1},y_{s+1}),\ p_l(x_m,y_{s+1}),...,\ p_s(x_d,y_d)$. It is important to note that $p_l(x_{m+1},y_{s+1})$ is not a processor in the network, thus this algorithm will not work for a fault on the right edge. This limitation comes from the observation that any messages travelling towards the right edge ($x_s<x_d$) will have to use the bypass connection around the faulty processor if the fault occurs such that the coordinates of the faulty processing element is $p_x(x_s,y_d)$. In this case we must be able to route to $p_x(x_{d+1},y_s)$.

Theorem 9

The routing algorithm is not complete if a fault exists on the right column of the network

Proof of Theorem 9

If we examine the path generated between source and destination processing elements in a reconfigured network with a single fault on the right edge of the network in the same row as the source processing element ($p_x(x_m,y_s)$)

$Q_p = p_s(x_s,y_s),\ p_l(x_{s+1},y_s),...,p_k(x_{d-1},y_s),\ p_k(x_{d+1},y_s),\ p_l(x_{d+1},y_{s+1}),\ p_l(x_{d+1},y_{s+1}),...,$

$p_s(x_d,y_d)$ $x_s<x_d$ and $y_s< y_d$

But $x_d = x_m$, $x_{d+1} \notin (P_o \cup P_f)$. Since the algorithm needs to route a message through a processing element not in the network. It can not work with a fault on the right edge.

■

We must address the problem of not allowing faults to occur in the final column of the network, for as the network size increases, it will become more likely that a fault will exists on the edge (in fact it will be almost inevitable). It has also been generally accepted that more faults occur on the edge of networks. Some possible approaches to handling faults at the boundary include:

1. Spare Processors at the boundary. It may be possible to have some spare processors at the boundary of a network. Reconfiguration could be applied to eliminate faults on the border. The disadvantage of this approach is that it requires redundant spare processors which will not normally be utilized.

2. It may be possible to eliminate the effect of faults at the border of a network by removing some functional processors from the network. Since faults will only affect messages destined for processors on the border, removal of some processing elements as valid destinations (but still able to route) at the border of the network will improve performance. This will be at the expense of harvest.

3. We will explore the use of additional connections in the next chapter which will be useful in increasing fault tolerance at the boundary.

Although the above techniques could be used to achieve fault tolerance on the border of a network, we will still have difficulties when multiple faults occur near the edge of a network. In general faults will have a much more significant effect when near the edge since there are less connections there, and thus less paths available to bypass faulty elements. This was noted in Chapter 4 when we noticed most routing delays were due to faults at the edge. In the next chapter we will be exploring the performance of these networks in the presence of multiple faults and interconnection failures. Since we will be extending the routing algorithms to multiple faults, we will be offering improved fault tolerance at the network boundaries.

5.5.2Hetro-XY Routing

In the previous section we examined a homogeneous routing algorithm. Although capable of routing any permutation in the presence of a single fault as long as the fault was not on the right border of the network, it does not provide good multiple fault coverage. This was an improvement over the other routing algorithm which was susceptible to faults anywhere on the border. In this section we show it is possible to construct a heterogeneous routing algorithm for reconfigured networks which can tolerate any single fault.

Consider the network shown in Figure 40. In this network we have shown a routing func-

tion which can route between any two processing elements in the network. This routing algorithm can be thought of as treating the two dimensional array as a linear array, and using a simple one dimensional routing algorithm to create a deadlock free routing algorithm. This particular algorithm would not be feasible to implement, as it would require some messages to travel a distance of n*m to travel from one corner to the other. It should also be noted that most of the vertical connections are not utilized.

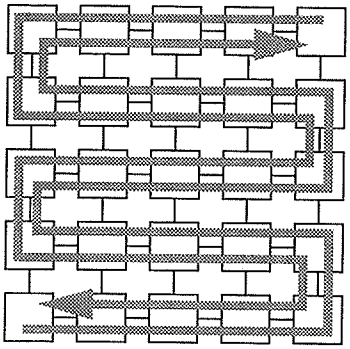


Figure 40 Heterogeneous Routing

While the above algorithm would be silly to implement it does offer insight into how to create a heterogeneous routing algorithm for a mesh. Notice that each of the eight possible ninety degree turns are implemented in the routing algorithm and yet it is deadlock free. The reason for this can be explained by considering that a message can travel on either of the two curves shown in Figure 40, but not on both. Each curve contains two turns from each cycle (clockwise and counter clockwise). Note that we could also add a one hundred and eighty degree turn at the upper right corner, and we would still have no cycles.

In order to improve the latency of this routing algorithm, we need to utilize the vertical connections which mostly are not used. To do this we add more turns to the routing algorithm, as we need a method for messages to change from the shown curves to the vertical connections. This can be accomplished by allowing the same turns on processing elements in the centre of the network, as those on the edge processing elements of the same row. This gives us two different routing func-

tions dependent upon which row (even or odd) the processing element is in. These turns can be

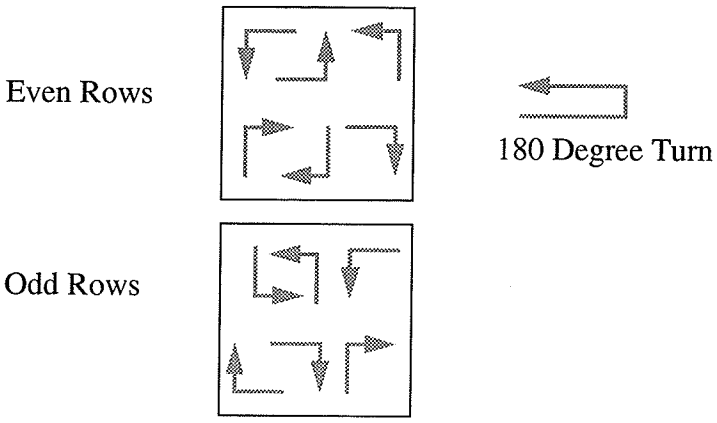


Figure 41 Turns in Even and Odd Row

seen in Figure 41. Zero degree turns are all allowed, and one hundred and eighty degree turns are permitted.

The routing function for this algorithm R_{HEO} is defined as follows:

R_{HEO}: C X P X M $\rightarrow$ C

Let c_I be the input channel

Let c_O be the output channel

Let $p_i(x_i,y_i)$ be the current processing element

Let $p_D(x_D,y_D)$ be the destination of message M

```
route_rheo(cI,pi,pD)
    IF (yD > yi) THEN
        cO = route_y(cI,pi,pD)
        IF (cO = NIL) THEN
            IF ((yD is ODD) AND (xD > xi)) OR ((yD is EVEN) AND (xD < xi)) THEN
                cO = route_y_over(cI,pi,pD)
            ELSE
                cO = route_x(cI,pi,pD)
    ELSE
        cO = route_x(cI,pi,pD)
        IF (cO = NIL) THEN
            IF ((yD is ODD) AND (xD > xi)) OR ((yD is EVEN) AND (xD < xi)) THEN
                cO = route_x_over(cI,pi,pD)
            ELSE
                cO = route_y(cI,pi,pD)
```

```
route_x(c_I,p_i,p_D)
        dx = x_D - x_i
        IF (dx = 0) THEN
            c_O = NIL
            RETURN
        IF (dx > 0) THEN
            c_O = c_f
            p_2 = destination(c_O)
            dx2 = x_D - x_2
            IF (dx2 < 0) THEN
                c_O = NIL
                RETURN
        ELSE
            c_O = c_b
            p_2 = destination(c_O)
            dx2 = x_D - x_2
            IF (dx2 > 0) THEN
                c_O = NIL
                RETURN
        IF NOT allowed[c_I,c_O] THEN
            c_O = NIL
        RETURN
route_x_over(c_I,p_i,p_D)
        dx = x_D - x_i
        IF (dx = 0) THEN
            c_O = NIL
            RETURN
        IF (dx > 0) THEN
            c_O = c_f
        ELSE
            c_O = c_b
        IF NOT allowed[c_I,c_O] THEN
            c_O = NIL
        RETURN
```

Theorem 10

The routing algorithm is complete in the presence of a single fault, not on the border of the mesh.

Proof of Theorem 10

Consider a fault at processing element $p_F(x_F,y_F)$, where p_F is not on the border of the net-

work. $0 < x_F < M-1$ and $0 < y_F < N-1$, where N and M are the size of the x and y dimensions.

We know from a previous theorem that if the source and destination are in the same row or column, then the routing algorithm can route between them, regardless of the number of faults. We will now consider the case where the source and destination are not in the same row or column.

We know that in the fault free case the following paths will be taken in routing from source processing element $p_S(x_S,y_S)$ to destination $p_D(x_D,y_D)$ such that $x_S \neq x_D$ and $y_S \neq y_D$.

$x_D > x_S$ and $y_D > y_S$

$$Q_P = p_S(x_S,y_S),\ p_a(x_S,y_S+1),\ldots,p_b(x_S,y_D),p_c(x_S+1,y_D),\ldots,p_D(x_D,y_D)$$

$x_D < x_S$ and $y_D > y_S$

$$Q_P = p_S(x_S,y_S),\ p_a(x_S,y_S+1),\ldots,p_b(x_S,y_D),p_c(x_S-1,y_D),\ldots,p_D(x_D,y_D)$$

$x_D > x_S$ and $y_D < y_S$

$$Q_P = p_S(x_S,y_S),\ p_a(x_S+1,y_S),\ldots,p_b(x_D,y_S),p_c(x_D,y_S-1),\ldots,p_D(x_D,y_D)$$

$x_D < x_S$ and $y_D < y_S$

$$Q_P = p_S(x_S,y_S),\ p_a(x_S-1,y_S),\ldots,p_b(x_D,y_S),p_c(x_D,y_S-1),\ldots,p_D(x_D,y_D)$$

If we know consider the path taken in the presence of fault p_F. We know that the fault can only affect the path taken if:

$x_F = x_S$ and $y_F = y_D$ for $y_D > y_S$

or

$x_F = x_D$ and $y_F = y_S$ for $y_D < y_S$

Therefore we consider the following cases:

$x_D > x_S$, $y_D > y_S$, $x_F = x_S$ and $y_F = y_D$ (y_D is an even row)

$$Q_P = p_S(x_S,y_S),p_a(x_S,y_S+1),\ldots,p_b(x_S,y_F-1),p_c(x_S,y_F+1),p_d(x_S+1,y_F+1),\ldots, p_e(x_D,y_D+1),\ p_D(x_D,y_D)$$

$x_D > x_S$, $y_D > y_S$, $x_F = x_S$ and $y_F = y_D$ (y_D is an odd row)

$$Q_P = p_S(x_S,y_S),p_a(x_S,y_S+1),\ldots,p_b(x_S,y_F-1),p_d(x_S+1,y_F-1),\ldots,p_e(x_D,y_D-1),p_D(x_D,y_D)$$

$x_D < x_S$, $y_D > y_S$, $x_F = x_S$ and $y_F = y_D$ (y_D is an even row)

$$Q_P = p_S(x_S,y_S),p_a(x_S,y_S+1),\ldots,p_b(x_S,y_F-1),p_d(x_S-1,y_F-1),\ldots,p_e(x_D,y_D-1),p_D(x_D,y_D)$$

$x_D < x_S$, $y_D > y_S$, $x_F = x_S$ and $y_F = y_D$ (y_D is an odd row)

$$Q_P = p_S(x_S,y_S),p_a(x_S,y_S+1),\ldots,p_b(x_S,y_F-1),p_c(x_S,y_F+1),p_d(x_S-1,y_F+1),\ldots, p_e(x_D,y_D+1),\ p_D(x_D,y_D)$$

$x_D > x_S$, $y_D < y_S$, $x_F = x_D$ and $y_F = y_S$ (y_D is an even row)

$Q_P = p_S(x_S,y_S),p_a(x_S+1,y_S),...,p_b(x_F-1,y_S),p_c(x_F+1,y_S),p_d(x_F+1,y_S-1),...,p_e(x_F+1,y_D), p_D(x_D,y_D)$

$x_D > x_S$, $y_D < y_S$, $x_F = x_S$ and $y_F = y_D$ (y_D is an odd row)

$Q_P = p_S(x_S,y_S),p_a(x_S+1,y_S),...,p_b(x_F-1,y_S),p_c(x_F-1,y_S-1),...,p_e(x_F-1,y_D), p_D(x_D,y_D)$

$x_D < x_S$, $y_D < y_S$, $x_F = x_S$ and $y_F = y_D$ (y_D is an even row)

$Q_P = p_S(x_S,y_S),p_a(x_S11,y_S),...,p_b(x_F+1,y_S),p_c(x_F+1,y_S-1),...,p_e(x_F+1,y_D), p_D(x_D,y_D)$

$x_D < x_S$, $y_D < y_S$, $x_F = x_S$ and $y_F = y_D$ (y_D is an odd row)

$Q_P = p_S(x_S,y_S),p_a(x_S-1,y_S),...,p_b(x_F+1,y_S),p_c(x_F-1,y_S),p_d(x_F-1,y_S-1),..., p_e(x_F-1,y_D), p_D(x_D,y_D)$

It can be seen that for all these paths, each processing element is in the network, and is not faulty. Since each path is valid, the routing algorithm can route around a single fault not on the border of the network.

■

Theorem 11

The routing algorithm is deadlock free.

Proof of Theorem 11

A wormhole routing algorithm can be shown to be deadlock free is we can find a ordering for connections in the network such that any path through the network will use strictly decreasing (or increasing) channel numbers[19].

We will use the numbering for channels shown in Figure 42. Notice that the channel num-

bers are different for even and odd rows. The channels are numbers so that the routing func-

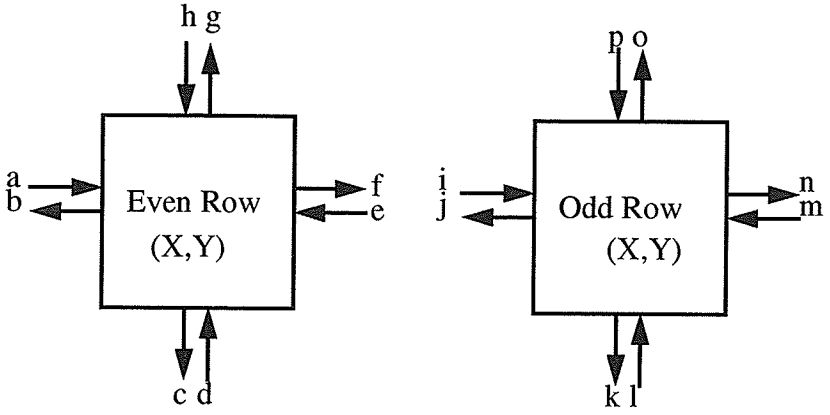


Figure 42 Channel Numbering

tion simply can select any output channel with a number less the input channel. The channel numbers a...p are defined as follows for a M by N array:

$a(X,Y) = f(X-1,Y) = 4(M+2)(N+2) -2(X+(Y+1)(M+2))$

$b(X,Y) = 2(X+1) + 2(Y+1)(M+2)$

$c(X,Y) = 2(X+1) + 2(Y+1)(M+2) +1 = b(X,Y) + 1$

$d(X,Y) = o(X,Y-1) = 4(M+2)(N+2) - 2(((M+2)-(X+1)) + Y(M+2)) + 1$

$e(X,Y) = b(X+1,Y) = 2(X+2) + 2(Y+1)(M+2)$

$f(X,Y) = 4(M+2)(N+2) -2((X+1)+(Y+1)(M+2))$

$g(X,Y) = 4(M+2)(N+2) -2((X+1)+(Y+1)(M+2)) + 1= f(X,Y) + 1$

$h(X,Y) = k(X,Y+1) = 2((M+1)-(X-1))+2(Y+2)(M+2)+1$

$i(X,Y) = n(X-1,Y) = 2((M+1)-X) + 2(Y+1)(M+2)$

$j(X,Y) = 4(M+2)(N+2) -2(((M+1)-(X+1)) + (Y+1)(M+2))$

$k(X,Y) = 4(M+2)(N+2) -2(((M+1)-(X+1)) + (Y+1)(M+2)) = j(X,Y)+1$

$l(X,Y) = g(X,Y-1) = 4(M+2)(N+2) -2((X+1)+(Y)(M+2)) + 1$

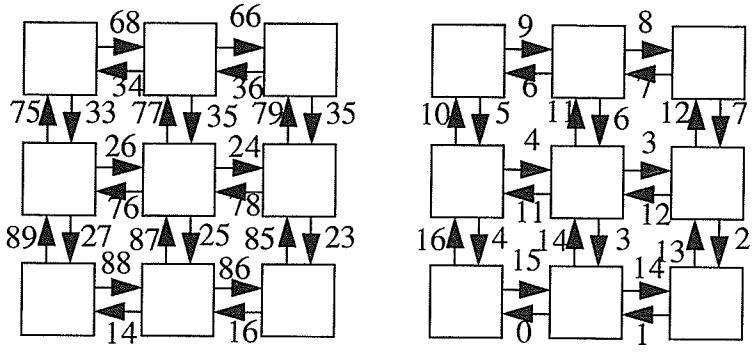
$m(X,Y)= j(X+1) = (M+2)(N+2) -2(((M+1)-(X+2)) + (Y+1)(M+2))$

$n(X,Y) = 2((N+1)-(X+1)) + 2(Y+1)(M+2)$

$o(X,Y) = 2((N+1)-(X+1)) + 2(Y+1)(M+2) + 1= n(X,Y)+1$

$p(X,Y) = k(X,Y+1) = 4(M+2)(N+2) -2(((M+1)-(X+1)) + (Y+2)(M+2))$

An example of channel numbering for a three by three mesh is shown in Figure 43(a). It is possible to come up with other numbers, including ones which will keep all numbers in the range 1..MN, and also ones which will number channels with non unique numbers. An example of this shown in Figure 43(b).



(a) (b)

Figure 43 Channel Numbering

Now, we need to show that each of the allowed turns represents moving from a channel with a higher number to a channel with a lower number.

Consider the following values:

$\alpha = 4(M+2)(N+2) - 2(M+N(M+2))$

$\beta = 2M + 2N(M+2)$

We can show that:

$\alpha = 4(M+2)(N+2) - 2(M+N(M+2)) = \beta + 4MN + 4M = 8N$

$\alpha > \beta$, since $M,N > 0$

Consider the Even Row allowed turns:

LD: $e > c$

$e = c+1$

RU: $a > g$

$a = g + 1$

UL: $d > b$

$d = 4(M+2)(N+2) - 2(((N+2)-(X+1)) + Y(M+2)) + 1 > \alpha$

$b = 2(X+1) + 2(Y+1)(M+2) < \beta$

Therefore $d > b$, since $\alpha > \beta$.

UR: $d > f$

$d = 4(M+2)(N+2) - 2(((M+2)-(X+1)) + Y(M+2)) + 1$

$f = 4(M+2)(N+2) - 2((X+1)+(Y+1)(M+2))$

$(N+2)-(X+1) < M + 1$

$(X+1) < M+1$

$(Y+1)(M+2) > Y(M+2)$

Therefore $d > f$, since $X,Y,M,N > 0$.

DL: $h > b$

$h = 2((M+1)-(X-1))+2(Y+2)(M+2)+1$

$b = 2(X+1) + 2(Y+1)(M+2)$

$((M+1)-(X-1)) > (X+1)$

$(Y+2)(M+2) > (Y+1)(M+2)$

Therefore $h > b$, since $X,Y,M,N > 0$.

RD: $a > c$

$a > \alpha$

$c < \beta$

Therefore $a > c$, since $\alpha > \beta$.

Consider the Even Row forbidden turns:

DR: $f > h$

$f > \alpha$

$h < \beta$

Therefore $f > h$, since $\alpha > \beta$.

LU: $g > e$

$g > \alpha$

$g < \beta$

Therefore $g > e$, since $\alpha > \beta$.

Similarly the Odd Row allowed turns can be shown to meet the following criteria:

DR: $p > n$

UL: $l > j$

DL: $m > k$

LU: $m > o$

RD: $i > k$

UR: $l > n$

And the Odd Row forbidden turns:

RU: o > i

DL: j > p

As well as the zero degree turns:

a > f, e > b, h > c, d > g

Since we have shown that it is possible to construct an ordering of channels, the algorithm is guaranteed to be deadlock free.

■

Once again this routing algorithm has the limitation of not being able to route in the presence of single faults on the boundary of the network. In addition to the techniques mentioned for the Modified XY routing algorithm, we could also utilize reconfiguration of the external connections to improve fault tolerance of the network. For example consider the network shown in Figure

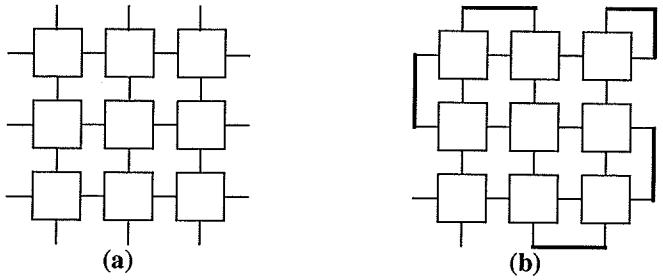


Figure 44 External Reconfiguration

44. With the addition of reconfigured links shown in Figure 44(b) it is possible to achieve fault tolerance for single faults on the border of the network.

5.6 Simulation Results

In this section we will examine some simulation results of the routing algorithms developed in this chapter. All the plots in this section are based on a 25 by 25 processor array. This size was selected as it represents a reasonable estimation of network size, and allows simulation in reasonable time.

5.6.1 Modified XY Routing

Figure 45 shows the probability of the simple XY routing algorithm being able to route for various yield rates.The curve labelled connections shows the percentage of paths between processors which successfully route messages. The second curve shows the percentage of processors who can successfully route to all other processors in the network. Despite the high path success rate, a large number of processors are not able to route to each other for high yield rates.

A similar curve is shown for the Modified XY routing algorithm in Figure 46. This routing algorithm offers clearly improved performance over the simple XY routing algorithm.

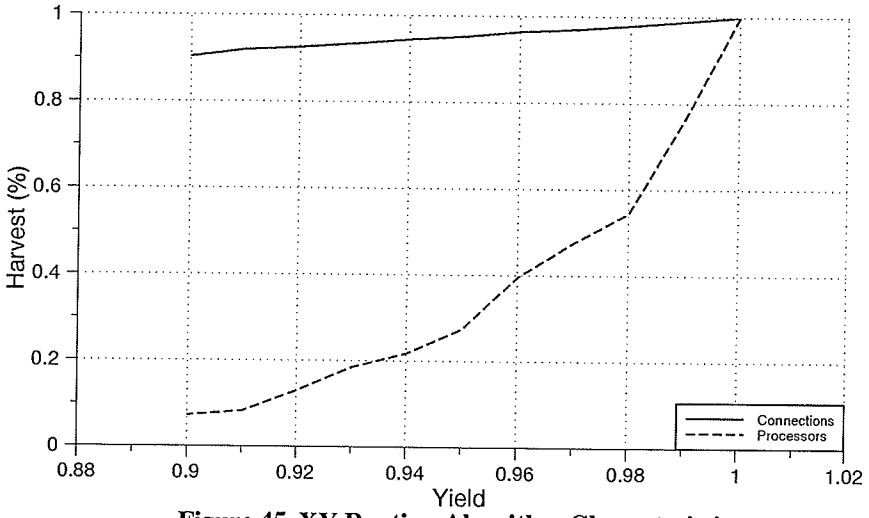


Figure 45 XY Routing Algorithm Characteristics

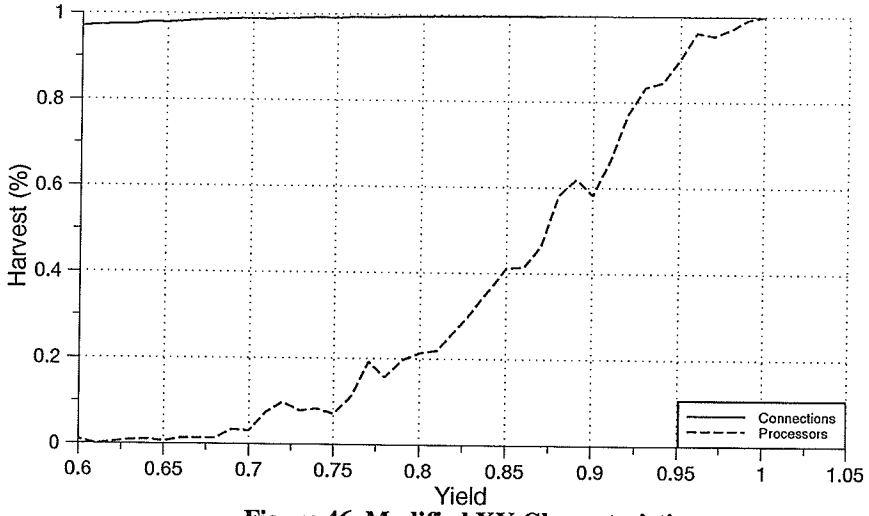


Figure 46 Modified XY Characteristics

5.6.2Heterogeneous Routing Algorithm

Figure 47 and Figure 48 show a comparison between the modified XY routing algorithm and the HetroXY routing algorithm. The modified XY routing algorithm offers slightly better fault tolerance than the HetroXY routing algorithm. This is primarily due to better fault tolerance at the border of the network. The Modified XY routing algorithm is highly susceptible to faults on the

right border of the network, whereas the HetroXY routing algorithm is susceptible to faults on all borders on the network.

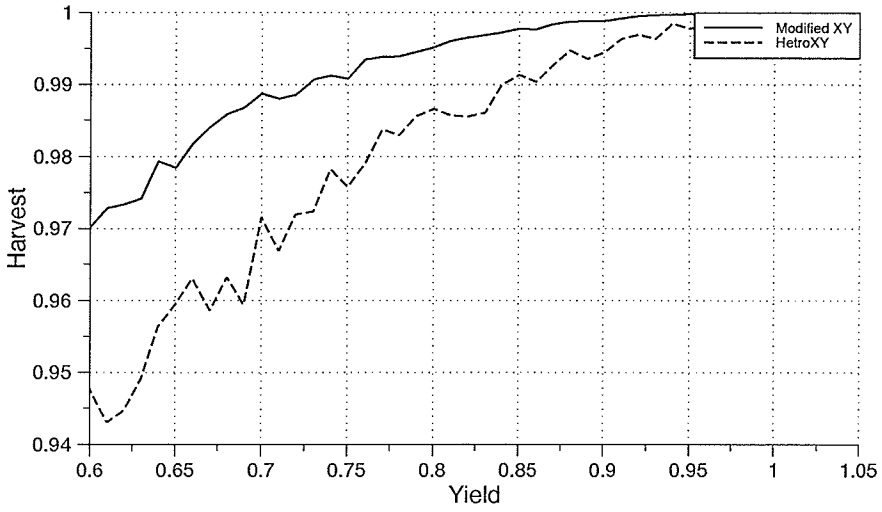


Figure 47 Paths in HetroXY

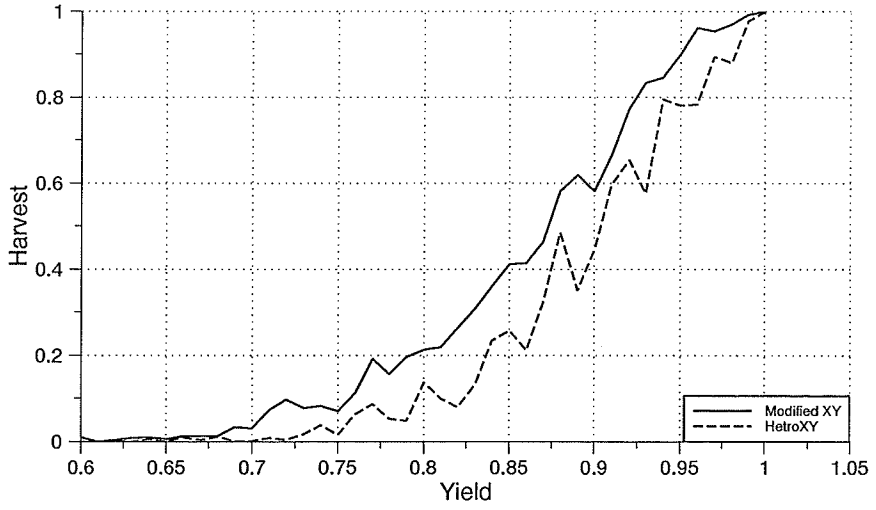


Figure 48 Processor harvest HetroXY

Figure 49 shows a plot of the average path length taken between processors which can successfully route to each other. In is interesting to note that the average path length decreases and the number of faults increases. This is due to the reconfiguration which will lower the diameter of the network as reconfiguration occurs. As more faults are bypassed, the separation between processors will on average decrease.

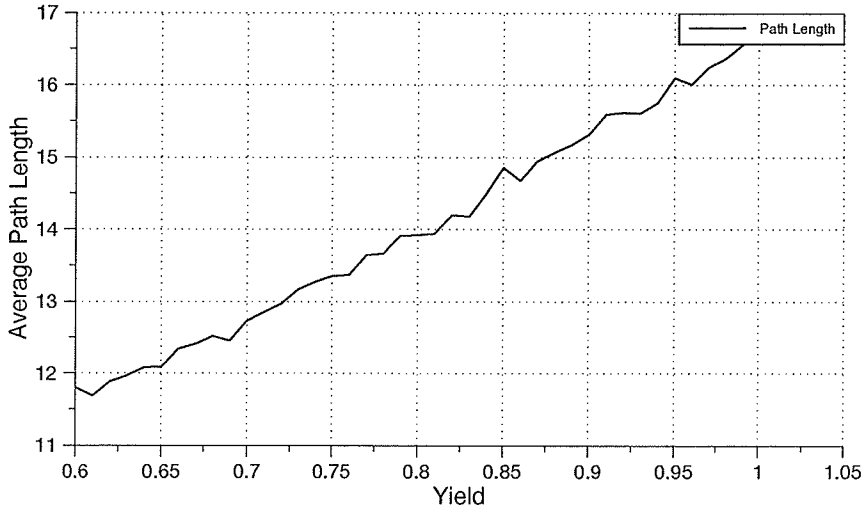


Figure 49 Average Path Length

5.7 Extensions to Hex Arrays

Although our interests in this thesis are in mesh connected networks, the results are extendable to other networks. In this section we will briefly look at extending the routing algorithms to processor arrays with six (hexagonal arrays) and eight nearest neighbours. These networks will require more complex routing algorithms and larger routing hardware to support the increased degree of the network, but can utilize the increased connectivity to improve network performance and fault tolerance.

Figure 50 shows a sample hexagonal network.

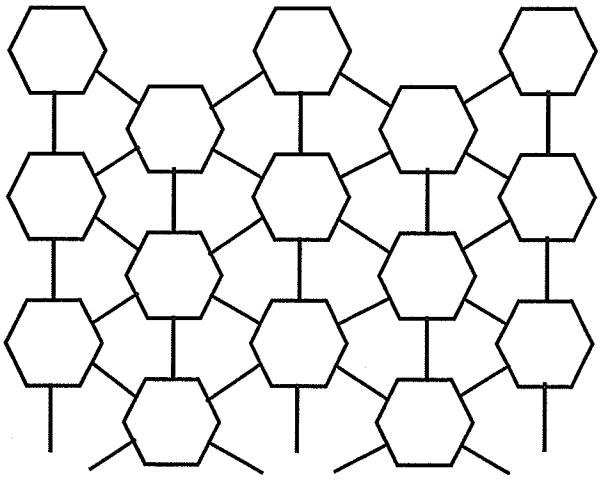


Figure 50 Hexagonal Array

An alternative representation of the network in Figure 50 is shown in Figure 51. When

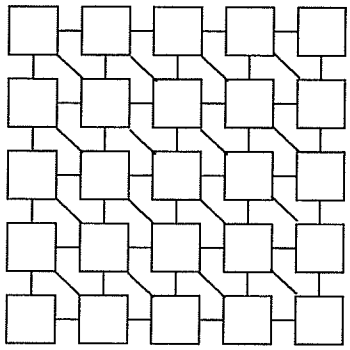


Figure 51 Hexagonal Array

shown this way we see that we can still utilize the Cartesian coordinate system for the addresses of the processors in the network. We now define the extended set of connections for a network to be the set of connections in the network, not in a mesh connected network.

> Definition 14: Let C be the set of connections in a mesh connected network. Let C_H be the set of connections in a network with the same processors as a mesh connected networks, and a superset of connections. We also define C^* to be the set C_H-C. Then we define the C^* to be the extended set of C.

Theorem 12

> Let c_i be an element of the extended set of connections for a network ($c_i \in C^*$) and let c_j be a member of the mesh set of connections for a network ($c_j \in C$). A routing algorithm R* which uses channels $C \cup C^*$ is deadlock free and contains the relation:
>
> $(c_j,c_i) \in R^*$
>
> if
>
> $\exists\ Q_P(\text{destination}(c_j), \text{destination}(c_i))$ such that $Q_P \in C$

Proof of Theorem 12

> We will prove this theorem by assuming that the connection c_i is in the deadlock-free routing algorithm R^*. We will show that it is possible to establish an ordering on the channels in the network such that each valid path follows channels of strictly descending order.
>
> We know that routing function R which operates on the mesh is deadlock-free. Hence it is possible to obtain a numbering of channels in C such that messages only take channels of smaller values than previously taken. We will denote this numbering by the function $CN_A(c_j)$. Likewise we will use a numbering on channels in the extended network $CN_B(c_e)$ where $c_e \in C \cup C^*$.
>
> We will also represent each channel from the extended set by two channels sequentially

placed (see Figure 52). We will refer to the two segments of c_i as c_{i1} and c_{i2}. This allows

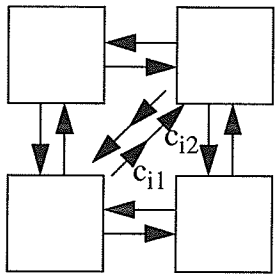


Figure 52 Channel Splitting

us to assign a unique channel number to each side of the channel c_i. We will maintain that $c_{i1} > c_{i2}$ so that the channel can always route messages.

Let $CN_B(c_e) = 2 * CN_A(c_j)$ for all $c_j \in C$.

We know that if R is complete for the fault-free case, we know there is at least one path from any two processors in the network. Therefore there is at least one path from source(c_i) to the destination(c_i) which utilizes only channels in C. Let the set Q_P* denote the set of paths from source(c_i) to the destination(c_i) which utilizes only channels in C.

Let c_k be the channel with the maximum CN_A value for all channels in Q_P*. Likewise let c_l be the channel with the maximum CN_A with destination processor destination(c_i).

Let $CN_B(c_{i1}) = 2 * CN_A(c_k) + 1$

and let $CN_B(c_{i2}) = 2 * CN_A(c_l) + 1$

Since we have obtained a channel numbering in which messages only traverse channels with descending values, we have shown the routing algorithm R* to be deadlock free. We know that a message on channel c_j can select c_i if there exists a path from c_j to destination(c_i) and (c_j,header(Q_P)) $\in$ R where Q_P is a path from (destination(c_j) to destination(c_i)).

■

Corollary 2.2

If R is connected, then each extended output connection is selectable from at least one input connection.

Theorem 13

Let c_i be an element of the extended set of connections for a network ($c_i \in C^*$) and let c_j be a member of the mesh set of connections for a network ($c_j \in C$). The following relation is allowed by the deadlock-free routing function R*:

(c_i,c_j) $\in$R*

if

$$\forall\, c_S \in C_{DP},\ (c_S, c_j) \in R$$

where

$$C_{DP} = \{c_S \mid \forall\, p_a \in p_\alpha,\ ((\text{source}(c_i), p_a), (p_a, \text{destination}(c_j)) \in R\}$$

Proof of Theorem 13

We will use the same channel value numbering used in Theorem12.

Let $CN_B(c_{i1}) = 2 * CN_A(c_k) + 1$

and let $CN_B(c_{i2}) = 2 * CN_A(c_l) + 1$

C_{DP} is the set of channels which are directly dependent upon c_j, which is the set of channels which can eventually route to channel c_j. Since all these channels will have channel number values greater than c_j, the theorem holds.

■

5.8 Summary

In this chapter we discussed the concept of two dimensional Diogenes reconfiguration on a two dimensional mesh. It was shown that this reconfiguration increased the connectivity of the networks such that improved fault tolerant routing algorithms could be implemented. Although the reconfiguration algorithm produces a disordered network, we were able to exploit the property of the network that all processing elements in the same row or column are connected by any routing algorithm.

Two routing algorithms were presented which attempt to exploit the connectivity of the network. The Modified XY routing algorithm is a homogeneous routing algorithm which uses the bypass connections to route around faults in the eastbound direction. This algorithm is shown to be fault tolerant for all single faults not on the boundary of the network.

A heterogeneous routing algorithm was also presented which is based on a zig zag routing scheme through a network. By modifying the routing algorithm for alternative rows, a routing algorithm was found which offered better fault coverage, although still only offering single fault tolerance for faults not on the border of the network.

Modifications to the routing algorithms so that the algorithms could route in hexagonal networks was also discussed and it was shown that the extended connections may be utilized in connected routing algorithms,

CHAPTER 6: Adaptive Routing and Defective Interconnects[1]

The routing algorithms developed in the previous chapter were developed to route in the presence of faults, however they were not adaptive. In this chapter will extend the result of the previous chapter and introduce adaptive routing algorithms for Diogenes reconfigured networks.

In the previous chapter we had also discussed routing in a network which contained only faulty processing elements. All interconnects were considered to be operational. This is one of the most common fault models used in modelling networks, but it is not realistic for very large networks. Traditionally interconnect faults have been ignored, since the size and complexity of these are normally much less than that of the processing elements. In a WSI environment however where we are dealing with a large network, one we wish to scale to arbitrary size, it is not realistic to ignore interconnect faults.

In this chapter we will focus on developing reconfiguration methodologies and routing algorithms for network which contain both faulty processing and connection elements. The ability to route in the presence of faulty connections is also closely related to a routing algorithm's ability to route in the presence of congestion. In this section we will examine the problem of adaptive routing, and develop new network structures, reconfiguration, and routing algorithms.

6.1 Interconnect Failures

Theorem 14

No local deadlock free wormhole routing algorithm exists for single connected mesh networks with a single faulty interconnect.

Proof of Theorem 14

Let us assume that we have a routing algorithm R which is local, complete, deadlock free and fault tolerant for all single connection faults.

Consider a processor in a mesh, with connections labelled as in Figure 53. We refer to connection a on processor i as a_i.

Consider two processors P_i and P_j in a network which are not adjacent (there are no connections between them). If there are only two sets of connections connected to these processors (as in corner processors), then we know that any message destined from P_i to P_j can use either of the output connections from P_i. If it could not, then the routing algorithm R would not be complete and fault tolerant as a single connection fault would make the algo-

1. Material from chapter 5 and this chapter has been accepted at the 1995 IEEE International conference on WSI

rithm incomplete. For example, a processor in the upper left corner of the mesh will only have c, d, e, and f connections. Only two of these are output connections (c and e). Any message destined from this processor to another must be able to utilize either connection c or e, or else a single fault on either c or e will make the routing algorithm incomplete. Likewise any message destined for this processor must be able to use either of the input connections.

We also know that any deadlock free routing algorithm will only route along connections of strictly descending (or ascending) order. This implies that the values of the output connections of P_i must be larger than the input connections of P_j. The inverse also applies.

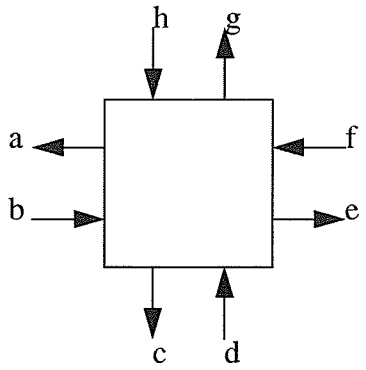


Figure 53 Connection Labels

Since we have four corner processors in a mesh, we have four processors with only two sets of connections. We refer to these four processors as A, B, C, and D (going clockwise from upper left). We know that

$$c_A,e_A > b_B,d_B,b_C,h_C,f_D,h_D$$

$$a_B,c_B > d_A,f_A,b_C,h_C,f_D,h_D$$

$$g_C,a_C > d_A,f_A,b_B,d_B,f_D,h_D$$

$$e_D,g_D > d_A,f_A,b_B,d_B,b_C,h_C$$

There are two general solutions to these equations:

One solution is where all output connections are greater than input connections:

$$c_A,e_A,a_B,c_B,g_C,a_C,e_D,g_D > d_A,f_A,b_B,d_B,b_C,h_C,f_D,h_D$$

and in one solution three of the processors have output connections with greater values than input, and the other solution has input connections greater than output. We assume processor D has input connections greater than output without loss of generality:

$$c_A,e_A,a_B,c_B,g_C,a_C > f_D,h_D > e_D,g_D > d_A,f_A,b_B,d_B,b_C,h_C$$

In either case we have at least three processors whose output connections have greater val-

ues than input connections. This means that these processors can not receive messages not destined for them, since they can not route from input connections to output connections.

Now consider the two processors connected to corner processor A. Both of these processors are along the edge of the network, and hence have three sets of connections. But we know that it can not route messages to the corner processor A, since it is not capable of forwarding messages. Since this only leaves two set of connections for it to use, it can be analysed the same as the corner processors, and hence it is not capable of forwarding messages. Since neither of the processors adjacent to A can forward messages, A can not route messages to processors not adjacent to it, and hence the algorithm is incomplete. This is a contradiction and thus proves the theorem.

■

The importance of the previous theorem is that it tells us that interconnect failures make things extremely difficult (if not impossible).

Redundancy is another possibility to handle connection faults. As with processor faults we wish to avoid the use of stand-by spares for connections for the following reasons:

1. Redundancy is wasteful unless the spares can be utilized even when there are not replacing faulty elements.

2. Depending upon the nature and location of a fault on a connection, reconfiguration may not be possible. For example a switch is always needed to select between a connection and its spare. A fault in this switch (still part of the connection) will prevent both connections from being utilized. As a result, it is possible that both a connection and its spare may be unusable.

Since it seems that we can not prevent or mask connection faults, we must develop algorithms to function in their presence.

There is also a similarity between a connection fault and network traffic. We mentioned in the previous chapter that we wish to implement adaptive routing, so that alternative paths may be utilized in a network when channels are busy. If we view a faulty connection as one that is busy for an infinite amount of time, we can see that by developing algorithms which are adaptive, we may be able to tolerate some connection faults. The similarity is between connection faults and busy connections is not perfect. The following difference should be carefully noted:

1. A busy connection will eventually become unbusy (available). Since we are implementing deadlock free and livelock free routing algorithms, a message can not become stuck forever. Thus a connection will always become available. However a faulty connection will remain faulty forever.

2. Most adaptive routing functions will select between available connections. If none are

available, the message will wait until one of the connections becomes free. With connection faults, none of the connections selectable by a routing function may be operational. This means a message may be undeliverable.

Since we have already shown that it is not possible to tolerant arbitrary connection faults, we will attempt to develop algorithms which will maximize the probability of being able to tolerate faults. Since it is safe to assume the probability of a faulty connection is quite small compared to the probability of a faulty processing element, this technique should prove successful.

6.2 Double X or Y Connections

One common technique to increase the connectivity of a mesh network is to double the number of connections in either the x or y directions for all nodes in the network. An example network for this is shown in Figure 54. This is a network with 2 pairs of connections between adjacent nodes in each row, and only a single pair of connection in the y direction.

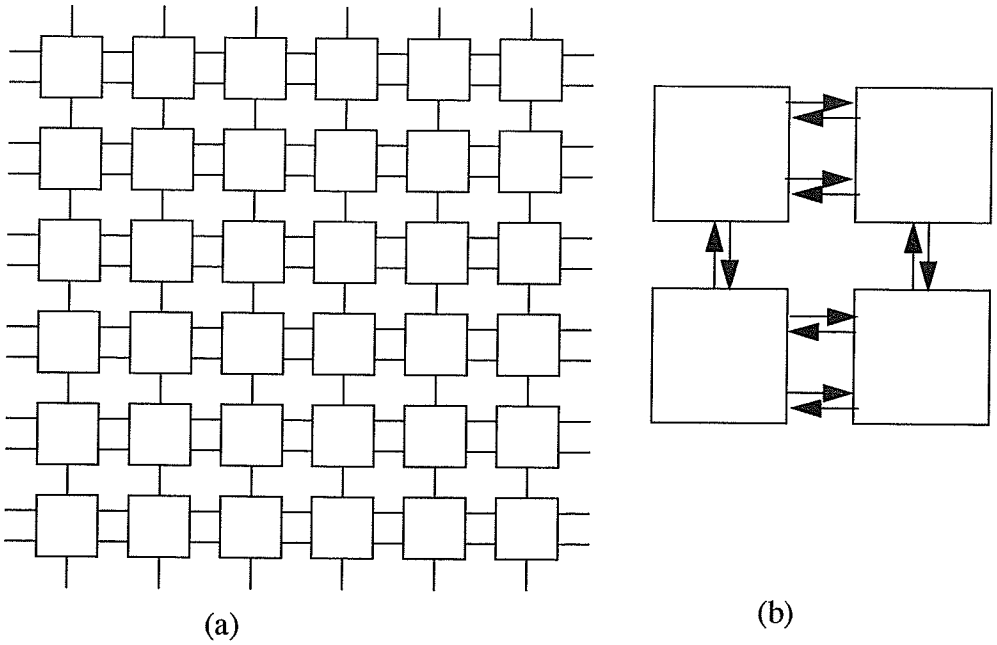


Figure 54 Double X Connections

Although the extra connections in the x direction provide extra connectivity, and allow greater flexibility and adaptability in routing, it is at an extra cost. An alternative to actually implementing the extra connections is to simply multiplex two set of connections together in a single set of connections. This allows two channels to exist, but requires only one connection. These techniques, referred to as virtual channels, still require that buffers and queues be implemented for each channel.

In this thesis we will assume a one to one correspondence between channels and connections. We will not utilize the virtual channel concept as we will require the extra connectivity of the physical connection to compensate for interconnect failures. Virtual connections would not offer added connectivity, as two channels would be rendered inoperative by a single fault.

6.3 Double XY Connections

It is possible to extend the concept of duplication of connections to both the x and y directions. An example network is shown in Figure 55.

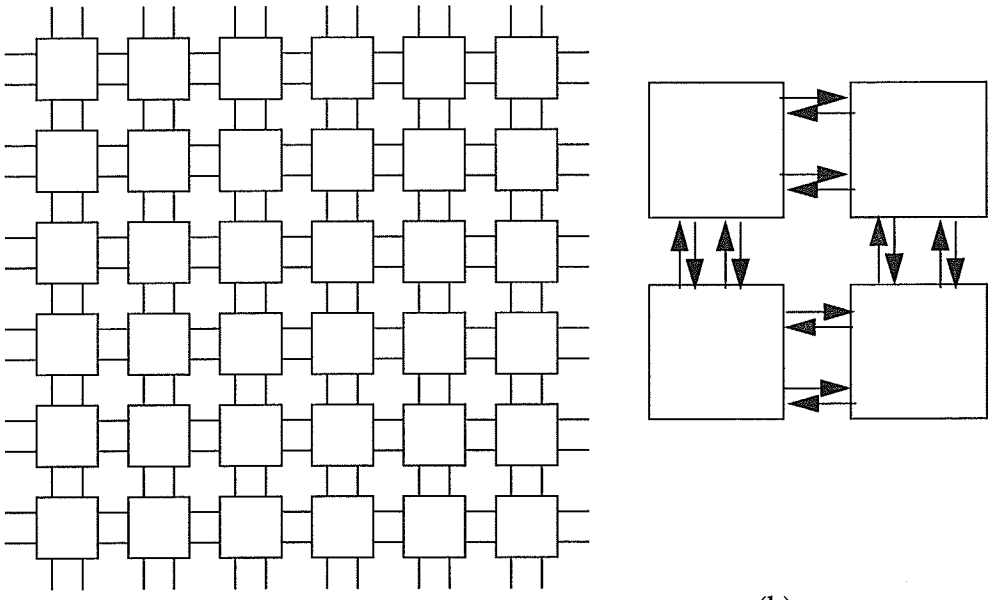


Figure 55 Double XY Connections

6.4 Connection Faults

We will make the following assumptions about connection faults:

1. Each connection fault is considered independent and only affects a single connection.

2. The number of connection faults will be significantly less than the number of processing element faults. This assumption is based on the observation that the physical space required for connections is significantly less than the physical space required for a processing element. Also the number of devices in a interconnect (buffers) is also significantly less than the number in a processing element.

3. Delay faults are not considered. These are faults which do not affect the functional characteristic of a connection, but simply the delay characteristics. Since we have assumed from the beginning that the network is asynchronous, delay faults can be tolerated.

4. A fault in an interconnect will prevent any reconfiguration of that connection.

One of the immediate implications of allowing connection failures is that the network may no longer be strongly connected.

6.5 Adaptive Routing

If we examine the shortest paths between two processors, we will in most cases find that there exist multiple paths of equal minimal distance. Figure 56 shows examples of different paths which may be taken from a source processor in the network to another processor whose x and y coordinates are greater than the source. Similar paths are available for other orientations of source and destinations. Since our routing functions use different behaviour at even and odd rows, we find that we can adaptively route at alternative rows. For the case shown in Figure 56, we can implement a partially adaptive routing function for messages destined in the north east direction (positive x and y directions). Likewise, the routing function will be partially adaptive for all combinations of source and destinations not on the same row or column. This offers a significant advantage over traditional homogeneous routing algorithms which will offer fully adaptive routing for two orientations of source and destination, and non adaptive routing for the other two orientations.

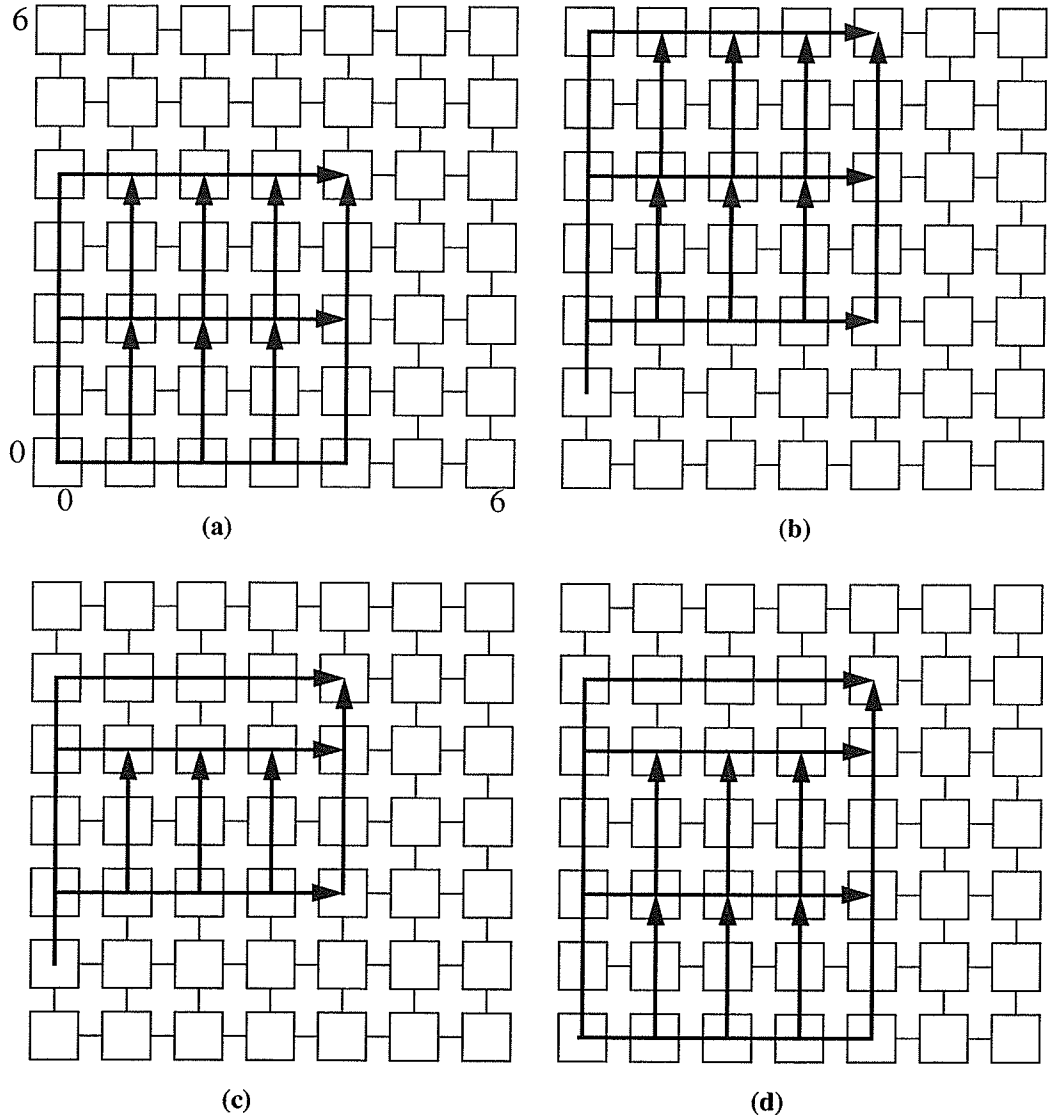


Figure 56 Adaptive Routing

We will also introduce a new routing algorithm at this time which is based on the heterogeneous routing algorithm presented previously. We will use this algorithm to create an adaptive routing algorithm. This routing algorithm will attempt to follow the curves shown in Figure 40 (Chapter 5) until the message is in the same column as the destination. Once there, the message will route directly to the destination.

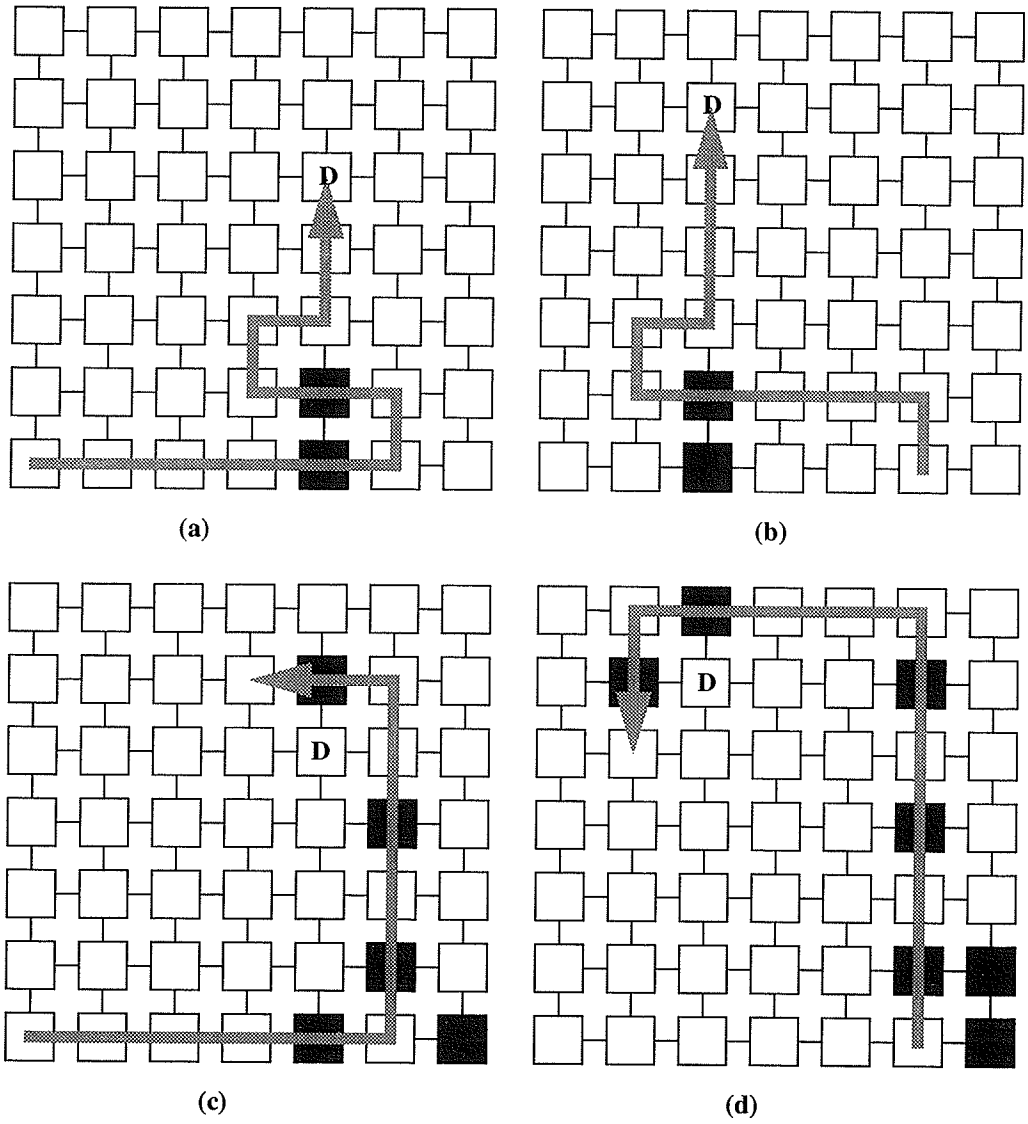


Figure 57 Fault Tolerant Routing

The routing function for this algorithm R_{HEO2} is defined as follows:

R_{HEO2}: C X P X M $\rightarrow$ C

Let c_I be the input channel

Let c_O be the output channel

Let $p_i(x_i,y_i)$ be the current processing element

Let $p_D(x_D,y_D)$ be the destination of message M

route_rheo2(c_I,p_i,p_D)

IF ($y_D = y_i$) **THEN**

c_O = route_y(c_I,p_i,p_D)

ELSE IF ($x_D = x_i$) **THEN**

c_O = route_x(c_I,p_i,p_D)

IF (y_i is EVEN) **THEN SWITCH**(p_i,p_D)

CASE(NORTH-EAST) c_O = route_x_over(c_I,p_i,p_D)

CASE(NORTH-WEST) c_O = route_y_over(c_I,p_i,p_D)

CASE(SOUTH-EAST) c_O = route_y_over(c_I,p_i,p_D)

CASE(SOUTH-WEST) c_O = route_x_over(c_I,p_i,p_D)

ELSE IF (y_i is ODD) **THEN SWITCH**(p_i,p_D)

CASE(NORTH-EAST) c_O = route_y_over(c_I,p_i,p_D)

CASE(NORTH-WEST) c_O = route_x_over(c_I,p_i,p_D)

CASE(SOUTH-EAST) c_O = route_x_over(c_I,p_i,p_D)

$\overline{\text{CASE}}$(SOUTH-WEST) c_O = route_y_over(c_I,p_i,p_D)

The route_y_over and route_x_over functions are the same as those in the HetroXY routing algorithm.

Two examples of the routing algorithm are shown in Figure 56(a) and (b) where the algorithm is able to route around faults. Figure 56(c) and (d) show cases where the algorithm is not able to route in the presence of faults.

The algorithm may be made adaptive by modifying the algorithm to take alternative paths in certain locations.

route_rheo2_adaptive(c_I,p_i,p_D)

c_O = NIL

IF ($y_D = y_i$) **THEN**

c_O = route_y(c_I,p_i,p_D)

ELSE IF ($x_D = x_i$) **THEN**

c_O = route_x(c_I,p_i,p_D)

IF (y_i is EVEN) **THEN SWITCH**(p_i,p_D)

CASE(NORTH-EAST)

c_O = route_x_over(c_I,p_i,p_D)

if (c_O = NIL) route_y_over(c_I,p_i,p_D)

CASE(NORTH-WEST)

c_O = route_y_over(c_I,p_i,p_D)

CASE(SOUTH-EAST)

c_O = route_y_over(c_I,p_i,p_D)

CASE(SOUTH-WEST)

c_O = route_x_over(c_I,p_i,p_D)

if (c_O = NIL) route_y_over(c_I,p_i,p_D)
ELSE IF (y_i is ODD) **THEN SWITCH**(p_i,p_D)
CASE(NORTH-EAST)
c_O = route_y_over(c_I,p_i,p_D)
CASE(NORTH-WEST)
c_O = route_x_over(c_I,p_i,p_D)
if (c_O = NIL) route_y_over(c_I,p_i,p_D)
CASE(SOUTH-EAST)
c_O = route_x_over(c_I,p_i,p_D)
if (c_O = NIL) route_y_over(c_I,p_i,p_D)
CASE(SOUTH-WEST)
c_O = route_y_over(c_I,p_i,p_D)

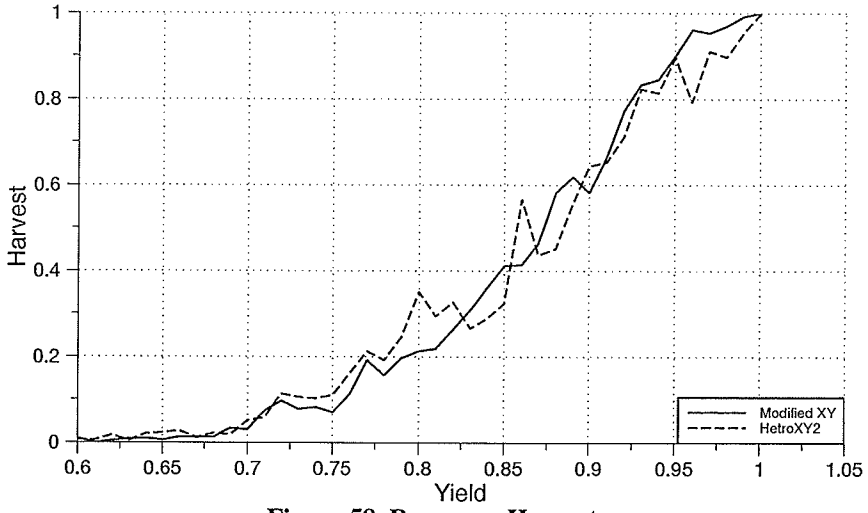


Figure 58 Processor Harvest

Figure 58 shows a comparison of this routing algorithm with the Modified XY routing algorithm presented in chapter 5. The results are comparable.

6.6 Cyclic Routing

Although the absence of cycles in the CDG guarantees that a wormhole routing algorithm is deadlock free, the presence of cycles does not always imply that a routing algorithm is subject to deadlock conditions. Three routing algorithms have been reported which utilize cycles in the CDG to increase the adaptiveness of a routing algorithm [20][55][68]. In this section we will investigate the properties of routing algorithms with cycles, establish a set of criteria in which these cycles may be utilized, and attempt to exploit these properties to increase both the adaptiveness and fault tolerance of a wormhole routing algorithm.

Definition 15: A routing function is said to be cyclic, if its channel dependen-

cy graph contains cycles.

In [20] it was reported that it was possible to construct a deadlock-free adaptive cyclic wormhole routing algorithm in which there exists multiple connections between processing elements. The channels in the networks are divided into distinct sets (two or more). The sets are classified as being either type A or type B. A routing function is defined for each set of channels. The routing function for the type A sets of channels may contain cycles, whereas the routing function for type B sets (typically there is only one) must be complete and acyclic. Routing is performed in a similar manner to any common routing algorithm for the topology along one of the sets of A channels. If a message is blocked, it is routed along a set of type B channels. These routing algorithms are deadlock-free.

Another cyclic routing algorithm was reported in [55]. The Message Flow Model (MFM) introduced the concept of deadlock-immune channels and networks, and exploited it to also increase the adaptiveness of a wormhole routing algorithm. It was also used very effectively for multicast routing. The MFM for unicast routing exploited the observation that certain channels could be used in a cyclic manner, as long as messages were not blocked on these channels. The MFM model was also applied to a network in which two disjoint sets of channels were available (the XY routing algorithm).

A third technique which exploits the same idea (called dynamic transitions) is also reported in [68]. Here they develop a new technique referred to as a Queue Dependency Graph, which can be used in the analysis of cyclic routing to determine deadlock conditions. Routing algorithms were developed for hypercubes, meshes, and shuffle exchange networks. A mesh algorithm was developed which uses 2 sets of channels in each direction between processing elements.

All three of these algorithms, although giving rise to different implementations, are based on similar observations that in limited circumstances, with modified routing approaches, routing in a deadlock-free manner with cyclic dependencies is possible. These algorithms were implemented in a slightly "ad hoc" manner, by which existing algorithms were modified to exploit the cycles where possible. In the remainder of this section we will attempt to develop a more complete theory of cyclic routing, and also attempt to exploit it to increase both the adaptiveness and fault tolerance of a network.

Both of the cyclic routing algorithms already investigated made use of a similar concept called deadlock-immune which was introduced in [55]. The key concept to developing cyclic algorithms is to recognize that the essential property of deadlock-free delivery of messages is to ensure that once a message allocates a channel, the message will also eventually release the channel. A message only releases a channel when the last flit has traversed the channel, and this can only occur when the message and all its flits are moving towards its destination, or being consumed at its destination. We now define the concepts of a channel being deadlock-free and deadlock-immune.

Definition 16: A channel c is said to be deadlock-free if any message which is waiting for it will eventually allocate the channel, and also will eventually free it.

Definition 17: A channel c is said to be deadlock immune for a message M if and only if once a message M allocates channel c, message M will eventually free channel c.

A channel c is also said to be deadlock immune for a message M if message M will never allocate channel c.

A channel c is said to be deadlock immune if it is deadlock immune for all messages M allowed by the routing function.

The essential difference between a channel being deadlock free and deadlock-immune is whether a message waiting for a channel can cause deadlock. In a deadlock immune channel, any message can allocate the channel, and it is guaranteed to release it eventually. Deadlock can occur if a message is blocked waiting for the channel. In a deadlock free channel, any message can allocate the channel, and any message can block waiting for the channel without creating a deadlock configuration. We illustrate this idea with the network shown in Figure 59. Four processors are shown (numbered 1 to 4) and eight channels (labelled A to H). The network shown in Figure 59(a) doesn't normally route message through the processing element in the upper left corner, as this would make the CDG cyclic as shown in Figure 59(b). Consider a message with source processor 2, and destination processor 4. Normally the message would be routed through channel F to processor 3, and then through channel G to processor 4. This will not create deadlock as it uses channels consistent with an acyclic CDG. The same message would not be allowed to travel through channel A to processor 1, and then through channel B to processor 4, as the routing function does not allow channel A to route to B. If we now consider the case where there is only one message in the network, we can route through channels A and B to the destination.

Now consider the case where we try to use channel A and B, but with other traffic in the network. Figure 59(c) shows the situation where a message a is attempting to traverse the path 2->1->4, and is blocked at processor 1 waiting for the channel B from 1->4 to become free. This channel will never become free since the message b using path 1->4->3 is blocked by message c using path 3->2->1, which is blocked by message a. This is a deadlock situation. Figure 59 shows an example where message a was not blocked at channel B (1->4). Since this message will eventually be delivered (since processing element 4 is its destination, and there is no means to block this message once a channel to processor 4 is allocated), it can never get in a deadlock situation; the other message b will eventually also be delivered (since it is waiting for a, which will be delivered). This example shows that it is possible to route messages using cyclic routing functions; however, we have to be careful about preventing deadlock waiting situations. A message can use a channel which creates a cycle but, if it does use it, it must be deadlock immune for that message, and it must not wait for it.

We will now develop a set of conditions which will be useful in creating routing functions which contain cycles.

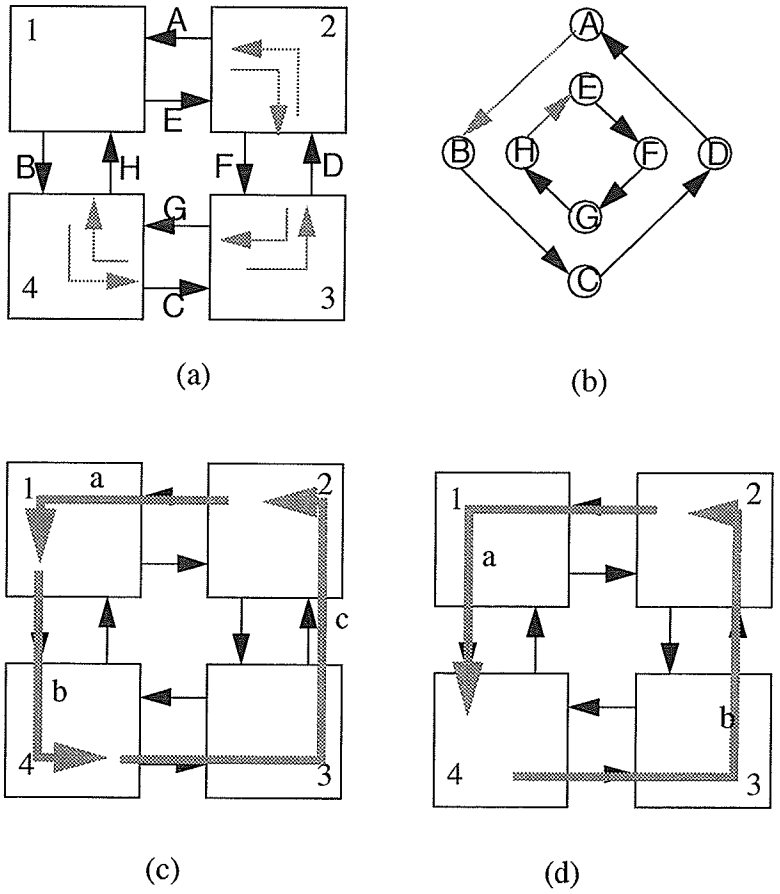


Figure 59 Cyclic Network

The following statements were shown in [55]:

1. A routing function is deadlock free if every channel is deadlock immune.

2. If the CDG for a routing function is acyclic, then every channel is deadlock immune.

It is also useful at this time to introduce a new classification for routing algorithms. We will refer to a routing function as separable if it possible to divide the channels in a network into disjoint sets, with a separate routing function for each set. A message routed using one routing function will only be allowed to allocate channels from the set associated with its routing function. We also define at the same time, the concept of a routing subfunction[20] which is the routing function for the subsets of channels.

> Definition 18: A routing subfunction R_1 of a routing function R, is a routing function which operates on a subset C_1 of the set of channels C of a net-

work. R_1 will be a mapping defined as such:

$$C_1 \, X \, p_C \, X \, p_D \rightarrow C_1$$

Definition 19: An extended routing subfunction R_1^* is a routing subfunction operates on an extended set to the subset associated with R_1.

$$C_1 \cup C_x \, X \, p_C \, X \, p_D \rightarrow C_1 \cup C_x$$

where C_x is the set of cyclic channels which can be used.

Definition 20: A set of channels C_1 is said to associated with routing subfunction R_1 if $C_1 \subset C$ and C_1 is the set of channels on which R_1 routes.

Definition 21: A routing algorithm is said to be separable if it is possible to split the routing function into multiple routing subfunctions, each of which operates with a disjoint subset of the channels in the network. A message must be able to be routed (in a fault free network) solely using one of the routing subfunctions. This implies that each subset of channels must span the network.

Definition 22: A routing function which can not be separated is called non-separable.

As an example of a non separable routing function consider the XY routing algorithm. It is not possible to separate the algorithm so that it meets the criteria of the definition. A modified XY routing algorithm with double Y connections is separable. It is possible to separate the routing algorithm into two disjoint algorithms, one routing east bound messages, and one routing west bound messages. North and south bound messages are handled by both. This can be seen in Figure 60.

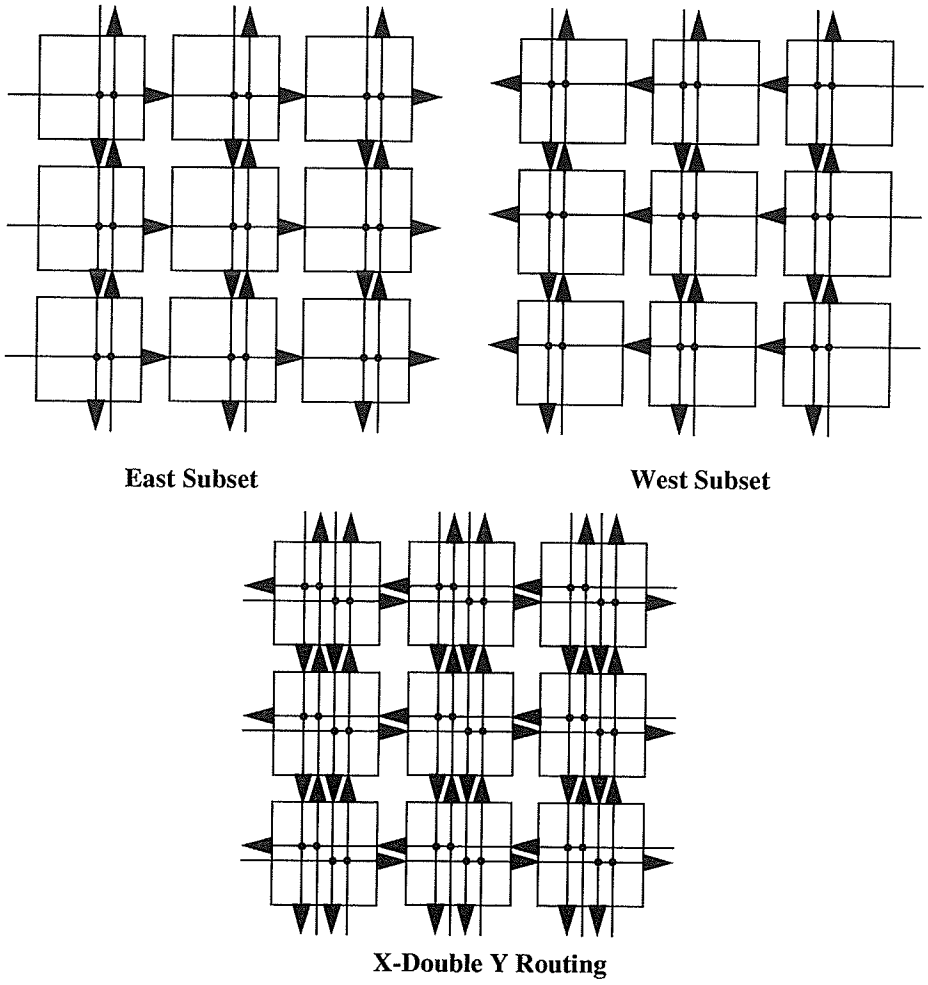


Figure 60 : Separation of X Double Y Routing Algorithm

One approach to introducing cycles into a routing algorithm, is to allow routing subfunctions to utilize channels outside of their associated channel set. We will refer to this as borrowing a channel. A channel will of course be borrowed from another channel subset.

Definition 23: A channel c is said to be borrowed by a routing subfunction R_1, if $c \notin C_1$, where C_1 is the channel subset associated with R_1.

If a channel is borrowed by routing subfunction R_i from another routing subfunction of R, then we have to ensure two things. First we must make sure that the borrowing of the channel, which may introduce cycles into the CDG, does not create a possible deadlock configuration. Secondly, we must also ensure that the borrowed channel may be used by the routing subfunction R_i in such a way that it does not create deadlock situations within R_i.

Definition 24: A channel c is said to be able to be utilized by a routing sub-

function R_i if it deadlock free for any messages routed by R_i.

It can be noted that if R_i is deadlock free, then all channels $c_i \in C_i$ are able to be utilized by R_i. We now define a property of a channel with respect to the extended routing subfunction which allows some channels to be used only if the messages do not wait for the channel to become available. This means that the messages may use the channel only if it is available when selected by the extended routing subfunction, otherwise it must use a channel which can be utilized, or wait on a channel which can be utilized.

Definition 25: A channel c is said to be able to be utilized without waiting by an extended routing subfunction R_i^* if it is deadlock immune for any messages routed by R_i but is not able to be routed by R_i with creating a deadlock configuration.

If we have a channel available to be utilized by a routing subfunction, we have to ensure that it is desirable to use it. In other words, just because a channel is available to be used, does not imply that it is worthwhile to use it. First we have to make sure that the utilization of this channel by a routing subfunction doesn't create a deadlock configuration.

A set of special channels with respect to a message M, whose destination is p_D, is the set of channels C_{SINK}, whose members are those channels with destination p_D.

Definition 26: A channel c is said to be a sink for message M, if $c \in C_{SINK}$, where C_{SINK} is define as $\{c_s | c_s \in C_{SINK}\}$.

$c_s \in C_{SINK}$ iff destination(c_s) = destination(M).

Theorem 15

Any channel c_z, which is a sink for a message M, may be utilized without waiting by any extended routing subfunction R_i for message M.

Proof of Theorem 15

Consider a message M whose destination is processing element p_D, and whose head flit is currently allocating channel c_i, such that destination(c_i) = p_C. We also assume that

$$p_C \in A(p_D)$$

We know since both p_C and p_D are neighbours, that there exists a connection c_z, such that:

$$c_z = (p_C, p_D)$$

Let us also assume that (c_i, c_z) is not in the routing function.

If channel c_z, which is a sink for message M, is free, and is allocated by message M, we know that the message M will be consumed at the destination(c_z) by the assumption stated earlier for wormhole routing. Since this message will be consumed, c_z is deadlock immune

for message M, and may be utilized without deadlock, as long as message M does not wait for the channel c_z.

■

Theorem 16

Consider a deadlock-free routing subfunction R_i and associated channel subset C_i. If a message M is currently allocating channel c_a as its head channel, then an extended routing subfunction routing can route to shared channel c_S without creating a deadlock configuration, if the following conditions are met:

1. The message M does not wait for channel c_S to become free.

2. There exists a channel c_b, $c_b \in C_i$, and $c_b \in$output_channel(destination(c_S)) and c_a is not directly dependent upon c_b.

3. There exists a path from c_b to destination(M).

Proof of Theorem 16

We will show that Message M is deadlock immune for channel c_S, and thus we know that if condition 1 is also satisfied, then the channel can be used (without waiting) without causing deadlock.

Let us assume that message M has allocated channel c_S. We will now show the message M will eventually be consumed at destination(M). This will show us that channel c_S is deadlock-immune with respect to message M.

We know from condition 2, that there exists an output channel c_b such that source(c_b) = destination(c_S). We also know that there exists a path from this channel c_b to the destination(M) using routing subfunction R_i. Since R_i is deadlock free, we can route message M through c_b to the destination(M) and that any message allocating c_b will eventually be consumed. If c_b is currently allocated, it will eventually become free since R_i is deadlock immune. Since message M, now allocating c_S it will eventually allocate c_b and will eventually be consumed at its destination. Thus channel c_S is deadlock immune for message M.

Since message M could be any message in the network, channel c_S is deadlock immune for all messages.

Therefore channel c_S can be used as a shared channel, without creating deadlock.

■

Both of the reported cyclic routing algorithms were implemented on networks in which separable routing functions were defined. We will show that it is possible to implement cyclic rout-

ing also in non separable algorithms.

6.6.1 Channel-Set Dependency Graph

If we are utilizing separable routing functions, it has already been shown that it may possible to improve the adaptability of such algorithms by introducing selected cycles into the CDG. In this section we will develop a model of separable routing functions, and determine restrictions on the location of cyclic channels.

Let us assume that it is possible to separate R into n distinct routing sub functions, R_1, R_2,..., R_n

Let C be the set of channels on G, the network. We can divide the channels into n disjoint sets of channels C_1,C_2,..., C_n, where C_i is the channel subset associated with R_i.

$$C = C_1 \cup C_2 \cup ... \cup C_n$$
$$C_i \cap C_j = \Phi \; \forall \; i,j \in 1..n$$

We now define a channel set dependency graph. A dependency will exist between channel sets if it is possible for routing sub functions associated with one set to borrow a channel from another set.

> Definition 27: The Channel Set Dependency Graph of a network with respect to the subsets C_1,C_2,..., C_n is a graph where each channel subset corresponds to a vertex in the graph. A directed edge exists between two vertices C_i and C_j (directed from C_i to C_j) if routing subfunction R_i associated with C_i can borrow a channel from C_j.

Theorem 17

> If a routing function R is deadlock free, and is separable, then all of its routing subfunctions are deadlock free.

Proof of Theorem 17

> Consider any two channels c_i and c_j such that $c_i,c_j \in C_k$. If $(c_i,c_j) \in R_k$, then $(c_i,c_j) \in R$. Similarly If $(c_i,c_j) \notin R_k$, then $(c_i,c_j) \notin R$.
>
> Similarly consider any two channels c_l and c_m such that $c_l,c_m \in C_k$. If $(c_l,c_m) \in R$, then $(c_i,c_j) \in R_k$. Similarly If $(c_i,c_j) \notin R$, then $(c_i,c_j) \notin R_k$.
>
> As a result the CDG of R_k will be a proper subset of the CDG of R. Since the CDG of R is acyclic, the CDG of R_k is also acyclic. Hence R_k is deadlock free.
>
> Since R_k was chosen randomly, all subfunctions of R will be deadlock free.

■

Theorem 18

If each of the routing subfunctions of a routing function R are deadlock free, and the CSDG is acyclic, then the extended routing function R* is deadlock free.

Proof of Theorem 18

Consider a message M_1 being routed using one of the routing subfunctions R_k and a second message M_2 (M1 ≠ M2) being routed using R_l ($R_k \neq R_l$). We will first show that it is possible to create a deadlock situation if there exists a cycles in the CSDG such that R_k uses a shared channel from C_l and R_l uses a shared channel from C_k. Then we will show deadlock can not occur if there is no cycles in the CSDG.

Let us assume that message M_1 is currently allocating channel $c_i \in C_k$with its header flit. Likewise let us assume that message M_2 is allocating channel $c_j \in C_l$ with its header flit. Let us also assume that message M_1 is allocating channel $c_a \in C_l$ with one of its data flits. Likewise let us also assume that message M_2 is allocating channel $c_b \in C_k$ with one of its data flits. If Message M_1 waits for channel c_b while allocating channel c_a, and message M_2 waits for channel c_a while allocating channel c_b deadlock will result. This can be seen in Figure 61. The CSGD for this example can be seen in Figure 62 which is clearly cyclic.

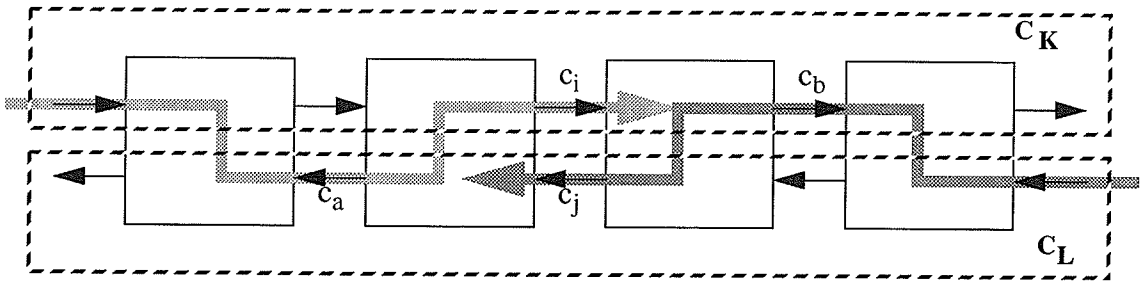


Figure 61 Deadlock from Cyclic CSDG

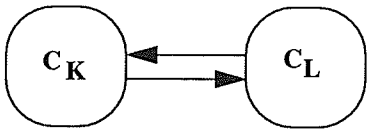


Figure 62 CSDG

We will now show that if the CSDG is acyclic, then the extended routing subfunctions will all be deadlock free. Consider a message M_1 using extended routing subfunction R_k and currently allocating channel $c_i \in C_k$with its header flit and allocating channel channels C_a ($\{c_a | c_a \notin C_k\}$ with some of its data flits.We will show that M_1 is deadlock immune for any paths to its destination using extended routing subfunction R_k.

Let us also consider any path Q_C which message M_1 will take in reaching its destination from its current channel. In order to ensure that M_1 is deadlock immune, we need to show that each channel in Q_C will eventually become free.

Consider any channel $c_m \in$ in Q_C. If c_m is allocated by a message M_2 using extended routing subfunction R_1 ($R_k \neq R_1$) then we know that M_1 will only be deadlock immune if M_2 is deadlock immune. Channel c_m will be dependent upon the channel that message M_2 is waiting for (if it is blocked). If we define set C_x as the set of indirect dependencies the set of all channels which c_m is dependent upon, it will be defined as follows:

$$C_x = \{c_k | c_k \in C_k \text{ and } c_m \text{ is dependent upon } c_k \text{ in the CDG for } R_k^*\} \cup$$

$$\{c_l | c_l \in C_l \text{ for } C_l \text{ such the } C_k \text{ is dependent upon } C_l \text{ in the CSDG}\}$$

Since there exists no $c_n \in C_x$ which is indirectly dependent upon c_m, then we know that M_1 is deadlock immune. Since all messages can likewise be shown to be deadlock immune, we know that the routing algorithm is deadlock free.

■

The previous two theorems establish a set of conditions on networks with separable routing functions in which each of the routing subfunctions are deadlock free. In these cases it is relatively easy to determine which cycles will not cause deadlock. It is also possible to implement routing algorithms in which not all the routing subfunctions are deadlock free.

6.6.2Nonseparable Cyclic Routing

Up to now we have been concerned with the borrowing of channels between routing subfunctions. If a routing algorithm is not separable, then it is not possible to separate it into distinct channel sets and associated routing subfunctions. Non separable routing algorithm can be viewed as routing functions which have only one routing subfunction. As we will see in this section, it is also possible to implement deadlock free cyclic non separable routing algorithms. We will demonstrate this concept with an example, consider a hypothetical routing algorithm which has a CDG graph as shown in Figure 63. Since we know we can always sort the channels of a deadlock free algorithm, it is always possible to lay out the channels in a horizontal row, and all dependencies will be shown as travelling from left to right.

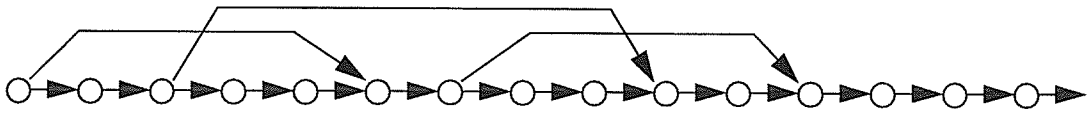


Figure 63 CDG of non separable routing algorithm

We now have to consider whether it is possible for a routing algorithm to utilize a channel which if utilized will create a cycle in the CDG. To do this we will first introduce a concept to channel borrowing, except it refers to channels which are borrowed within a single routing subfunction.

> Definition 28: A shared channel is a channel in a routing subfunction which introduces a cycle into the CDG when utilized without waiting by a processing element.

Theorem 19

> A complete deadlock-free routing subfunction may utilize a shared channel c_S for a message M if the last channel c_j allocated by M has index lower the c_S index(c_S) > index(c_j).

Proof of Theorem 19

> This theorem can be proved by utilizing Theorem 16. The index is the value associated with a channel, whereby deadlock free routing algorithms route by selecting channels with strictly decreasing indexes.
>
> We know if the index of the channel associated with c_j is lower than the index associated with c_S then c_j is directly dependent upon c_S. We also know that since the routing algorithm is complete, that there exists a path between each pair of processors. Thus the conditions of Theorem 16 are met, and the algorithm is deadlock free.

■

6.6.3Slow Routing

The theory developed in the previous sections for cyclic routing assumes that it is only possible to routed along shared channels when the channel is freed, and we are not allowed to wait on the channel. We can extend the flexibility of cyclic routing by allowing waiting on shared channels, but only for finite periods of time. We will refer to this as slow routing, as a message may wait for a arbitrarily long (but not infinite) period of time.

One of the easiest ways to implement slow routing is to use a repeated routing function. This will allow the routing function to be repeatedly applied until an available channel is found. It has the advantage over a single application of the routing function in that shared channels may be

attempted a few times before an available channel is found.

6.6.4Implementing Routing Algorithms

It is relatively straightforward to apply the theory of cyclic routing functions and to implement a routing function suitable for WSI. We know that if we have a complete deadlock-free routing subfunction and we have other routing subfunctions (which may not be deadlock free or complete), we can implement a complete deadlock-free adaptive routing algorithm by simply utilizing the complete deadlock-free routing subfunction as the primary routing function, and adaptively using the other routing subfunctions when congestion occurs in the network. The important property is that we only need one complete deadlock-free routing subfunction to base the algorithm on.

If we are implementing a routing algorithm for networks which contain multiple X and/or multiple Y channels, the different sets of connections form a natural boundary for separating routing subfunctions. Specifically, each set of X and Y channels can be used to implement a routing subfunction. We just have to ensure than one of them is complete and deadlock-free.

6.7 Routing

We are still faced with the problem of trying to implement a complete routing algorithm for a defective mesh. In chapter 4, we considered a non deterministic routing algorithm which would eventually deliver all packets, but the time taken may be unbounded. In chapter 5 we looked at wormhole routing, in which lower latency is possible; however, the network is much more susceptible to deadlock. In this chapter we have added communication faults which further complicate routing algorithms.

It can already be noted that in all routing algorithms developed for defective networks, some penalty is paid for fault tolerant routing. Although algorithms exist which are minimal in the presence of a small number of defects (routing on defective hypercubes with a smaller number of faults than dimension), all routing algorithms will become non minimal when dealing with an unspecified number of faults. Each time a non minimal routing algorithm routes a message, which is not on the shortest path, it is allocating additional resources, which block other messages, and hence slow down the network.

Definition 29: A processing element $p_j(x_j,y_j)$ is said to be similarly oriented to $p_i(x_i,y_i)$ with respect to $p_D(x_D,y_D)$ if

$$\frac{x_D - x_i}{|x_D - x_i|} = \frac{x_D - x_S}{|x_D - x_S|} = \frac{x_i - x_S}{|x_i - x_S|},$$

and

$$\frac{y_D - y_i}{|y_D - y_i|} = \frac{y_D - y_S}{|y_D - y_S|} = \frac{y_i - y_S}{|y_i - y_S|}$$

Definition 30: A processing element $p_j(x_j,y_j)$ is said to be traversable for $p_i(x_i,y_i)$ with respect to $p_D(x_D,y_D)$ if

Let Q_A^* be the set of off all paths from p_i to p_j

Let Q_B^* be the set of off all paths from p_j to p_D

Let C_A be the set of channels $\{c_A\}$ such that

destination$(c_A) = p_j$

$c_A \in Q_A^*$

and let C_B be the set of channels $\{c_B\}$ such that

source$(c_B) = p_j$

$c_B \in Q_B^*$

Then $p_j(x_j,y_j)$ is traversable if $(c_A,c_B) \in R$ for all $c_A \in C_A$ and $c_B \in C_B$

Theorem 20

A minimal adaptive routing algorithm R is complete if for every two processing elements $p_S(x_S,y_S)$ and $p_D(x_D,y_D)$, $p_S \neq p_D$, there exists a processing element $p_T(x_T,y_T)$ which is traversable with respect to p_D which is similarly oriented and closer by one hop in one direction to the destination.

Proof of Theorem 20

This theorem will show that any processor can always route to any other processor (assuming they are both functional) as long as there exists a processing element which we can route through to the destination, which is closer in one direction than the source.

We will without loss of generality assume that we wish to route closer in the Y direction.

Consider a source processing element $p_S(x_S,y_S)$ and a destination processing element $p_D(x_D,y_D)$. We know that we can always route to a processing element $p_{T1}(x_{T1},y_{T1})$, such that $y_{T1} = y_D + 1$. We also know that $(x_D-x_{T1})/|(x_D-x_{T1})| = (x_D-x_S)/|(x_D-x_S)|$. By recursion, we will always be able to route through a processing element $p_{TN}(x_{TN},y_{TN})$, $N = |y_D-y_S|$, $y_{TN} = y_D$. This processing element will also be traversable, and will be able to route to $p_D(x_D,y_D)$.

6.7.1Example Algorithms

The simplest example of this concept is routing in the fault-free mesh using a zig-zag algorithm. A message with source processing element $p_S(x_S,y_S)$ and a destination processing element

$p_D(x_D,y_D)$ will route towards the destination, avoiding the row and column of the destination processing element. The algorithm will eventually route to $p_T(x_T,y_T)$, where $|x_T-x_D| = 1$, and $|y_T-y_D| = 1$. From here the algorithm will route to the destination.

For the reconfigured networks we are looking at, we will exploit the HetroXY2 routing algorithm presented earlier in this chapter. This algorithm is capable of routing towards the destination processing element in the Y direction, and will be similarly oriented towards the destination processing element if we introduce a coordinate transform on odd rows to reverse the X coordinates. This algorithm will work as long as there is a path between alternating rows.

6.8 Reconfiguration and Routing Using Links

In the previous section we should note that it is possible to implement a complete routing function, as long as we can find a traversable processing element closer in the Y direction. This was based on the assumption that the network contained no connection faults. If we wish to consider connections faults, we must be able to guarantee that we can always route between processors in both the X and Y directions. In this section we will discuss the use of reconfiguration of channels to bypass faulty connections.

Definition 31: A link is a set of channels which are configured to replace a faulty connection.

Let C_L be the set of connections in a link.

$C_L = \{c_1, c_2,..., c_n\}$

$source(c_i) = destination(c_{i-1})\ 2 \leq i \leq n.$

Let c_f be the faulty connection which is replaced by link C_L.

$source(c_f) = source(c_1)$

$destination(c_f) = destination(c_n).$

An example of reconfiguration links is shown in Figure 64. In this example a double X and double Y connected network is implemented. The double connections are implemented using virtual channels, so that all channels between adjacent processing elements are using the same physical connection. This implies that when a single fault occurs, multiple channels must be reconfigured using links (shaded arrows in Figure 64).

The following observations may be made:

1. Only one set of channels are reconfigured using links in Figure 64. This is because the second set of channels is used as spares. We note that the links, which consist of three channels, utilize both X and Y direction channels. As a result of this reconfiguration, neighbour-

ing processors have only one set of channels in the X or Y direction. As a result, there is no need to reconfigure both sets.

2. When a channel is being used as part of a link, it is not allowed to be used as a regular channel simultaneously. This may create a deadlock situation.

3. Any message which enters the first channel of a link must exit the last channel of a link.

4. At least one set of channels per physical connection must not be used for link. This guarantees that we can implement a complete routing subfunction.

5. The channels not used as links, and not as the primary set of channels per connections, may be used adaptively by the extended routing subfunction in a cyclic manner.

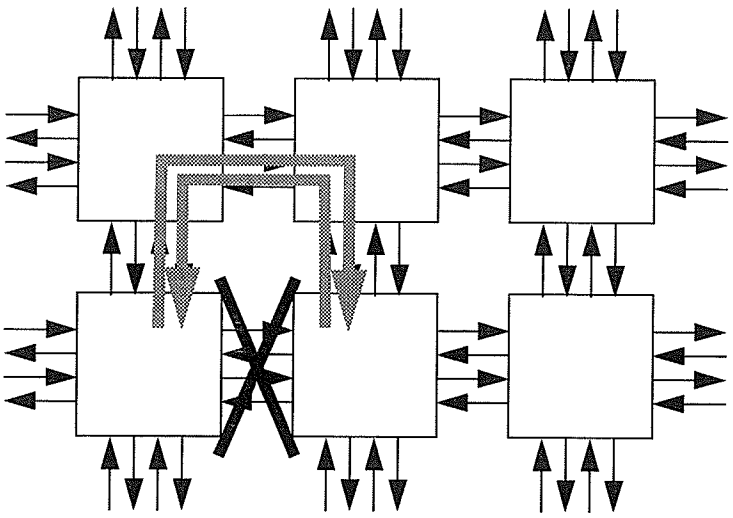


Figure 64 Link Reconfiguration

We implemented a very simple algorithm for determining the link paths to reconfigure a faulty connection. The set of shortest paths were found using a breadth-first search. A nondeterministic greedy algorithm was then used to select one of the paths, and it was chosen for the link. This algorithm is by no means optimal, but it works satisfactorily for the small distances involved in the links. Because of the presence of faulty processors, and the disordered nature of the network, more straightforward algorithms are difficult to implement.

Figure 65 shows a plot of the maximum number of links per connection required by link reconfiguration of a 25 by 25 processor array with a single connection between adjacent processors. A network will require this number of virtual channels plus one (for the routing function). It can be seen that for a low number of connection faults we would only require two virtual channels.

For networks with a large number of connection faults, we may require 4 virtual channels. In the case where we have no processor faults, we will require 3 virtual channels.

Figure 66 shows a plot for the same networks, except that we are plotting the average number of link channels used per connection. It can be seen that typically each connection will be utilizing approximately one link channel.

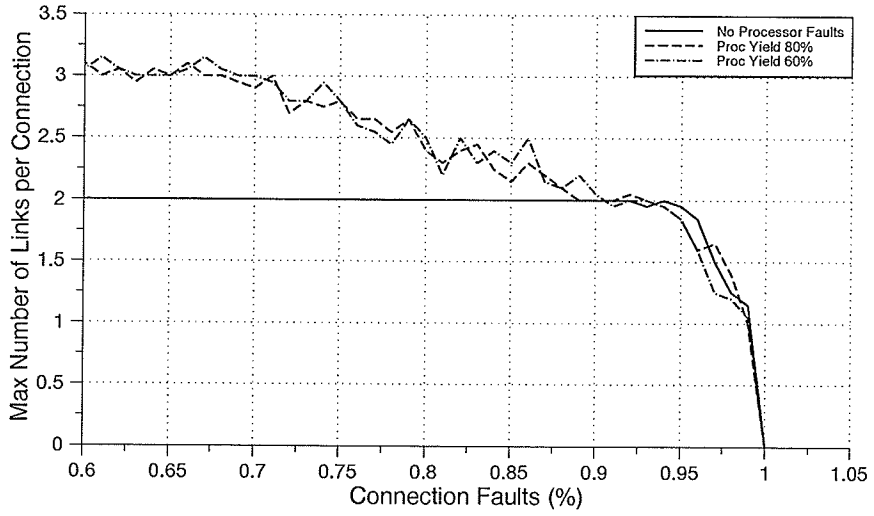


Figure 65 Links Per Connection

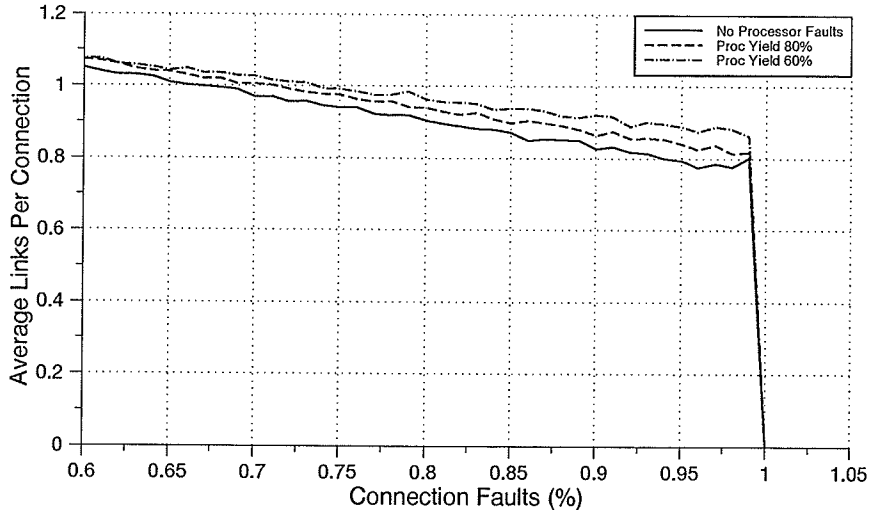


Figure 66 Average Links per Connection

6.9 Summary

In this chapter we performed the following:

1. Extended the results of the previous chapter for use with networks with multiple virtual channels.

2. We looked at adaptive routing algorithms for Diogenes reconfigured networks.

3. Examined routing with cycles in the CDG. This allows for more effective adaptive routing algorithms to be implemented.

4. We investigated using links to reconfigure networks with connection faults.

CHAPTER 7: Conclusions

7.1 Conclusions

In this thesis we have analysed the characteristics of WSI processor arrays from the perspective of disordered arrays, and dave tried to exploit the physical characteristics of these networks in order to implement a general purpose processing environment. While we did not implement any systems, we did investigate an important aspect of these systems: communications between processors. We have focused our attention on the problem of implementing routing algorithms which can successfully route in the presence of a large number of faulty elements. We have introduced numerous routing algorithms, and have shown some of the characteristics of the different approaches. While substantial progress has been made, there is still need for more research in these areas. What we have learned is that it can be done.

Percolation theory was found to be a useful tool in the analysis of processor array networks, and is capable of giving us bounds on the physical properties of these networks. We found for mesh architectures, that we need at least 60% yield to achieve connectivity between functional processors, and nearly 100% yield to achieve sufficient conductivity to allow message flow. In addition to these bounds, percolation theory gave us valuable insight into the physical characteristics of these networks, and these characteristics were verified when a routing algorithm was implemented for these networks.

We also developed a nondeterministic routing algorithm, which will route in the highly disordered networks of the WSI environment. This algorithm, based on biased random walkers, could successfully route between any connected processors in the network, although the network latency was unacceptable for yield rates near the percolation threshold. It was possible to improve this algorithm by adjusting the weights, so that messages would flow away from faults and congestion.

We learned from percolation theory that a network with faults will not have sufficient connectivity to achieve good message flows. With this in mind, we applied two dimensional Diogenes reconfiguration to the network, which improved network connectivity, and allowed for more reasonable network flow. The reconfiguration achieved almost 100% harvest of functional processors, but did not produce an ordered array. We developed routing algorithms for these types of networks, based on the wormhole routing methodology.

In order to improve the routing algorithms we extended the algorithms so that they could be adaptable. In an attempt to achieve fault tolerance for connection faults in the network, we also incorporated reconfigured links, and developed a method whereby these links configured from virtual channels. The spare virtual channels can be used by cyclic routing techniques also investigated in this thesis.

In the end, it was found feasible to implement routing in a highly defective environment such as those found in WSI systems.

7.2 Future Work

This thesis by no means solves all of the problems associated with WSI, or even those associated with routing in defective processor arrays. Some of the future areas for work include:

1. Implementation of some of the Routing Algorithms.

2. Alternative Reconfiguration Schemes.

3. New Routing Algorithms.

In addition to the work presented here, some more research will need to be conducted on issues relating to utilizing disordered processor networks. Research will need to investigate the following:

1. Mapping processes to reconfigured processor arrays.

2. Developing parallel algorithms for large networks.

References

[1]E. Aarts, J. Korst, *Simulated Annealing and Boltzmann Machines*, Wiley, 1989.

[2]A.H. Anderson, J.I. Raffel, P.W. Wyatt, "Wafer-Scale Integration Using Restructurable VLSI", IEEE Computer, April 1992, pp. 41 - 47.

[3]D. Badouel, C.A. Wuthrich, E.L. Fiume, "Routing Strategies and Message Contention on Low-dimensional Interconnection Network", University of Toronto Technical Report, Dept of Computer Science, 1991.

[4]K. Bharath-Kumar, J.M. Jaffe, "A New Approach to Performance-Oriented Flow Control",IEEE Transactions on Communications, Vol. COM-29, No. 4, April 1981, pp. 427-435.

[5]D.C. Blight, R.D. McLeod, "Performance Simulation of Wafer Scale Processor Arrays", CCECE, 1990.

[6]D.C. Blight, R.D. McLeod, "Transport Modeling of Wafer Scale Processor Array Dynamics", Canadian Conference on VLSI, pp. 6.7.1 - 6.7.8, October 1990, Ottawa Canada.

[7]D.C. Blight, R.D. McLeod, "Reconfiguration and Routing in Defective WSI processor Arrays", to be published in the seventh annual IEEE International Conference on WSI.

[8]D.C. Blight, R.D. McLeod, "An Adaptive Message Passing Environment for Wafer Scale Systems", IEEE Transactions on VLSI Systems, Vol 1., No. 4, December 1993.

[9]D.C. Blight, R.D. McLeod, "Adaptive Nondeterministic Routing in Mesh Connected Networks", Microelectronics Journal, Vol. 23 (1992), pp. 517-521.

[10]D.C. Blight, R.D. McLeod, "Nondeterministic Adaptive Routing Techniques for WSI Processor Arrays, IEEE International Workshop on Defect and Fault Tolerance, pp. 177-184, Dallas Texas, Nov. 1992.

[11]Blumen, A., J. Klafter, B.S. White and G. Zumofen, *Continuous-Time Random Walks on Fractals*, Phys. Rev. Lett., V. 53, 1984, pp.1301-1304.

[12]R.R. Boorstyn, A.Livne, "A Technique for Adaptive Routing in Networks", IEEE Transactions on Communications, Vol. COM-29, No. 4, April 1981.

[13]G. D. Burns, "A Local Area Multicomputer", Proceeding of the Fourth Conference on Hypercubes, Concurrent Computers, and Applications, 1989.

[14]R. O. Carlson, C.A. Neugebauer, "Future Trends in Wafer Scale Integration", Proceedings of the IEEE, Vol 74, No. 12, December 1986.

[15]G.H. Chapman, M. Parameswaran, M. Syrzycki, "Wafer Scale Transducer Arrays", Computer, Volume 25, No. 4, pp. 50-56 April 1992,

[16]G.H. Chapman, J.M. Canter, and S.S. Cohen, "The Technology of Lasser-Formed Interactions for Wafer-Scale Integration", Proc. Int'l Conference on Wafer Scale Integration, IEEE Computer Soc Process. 1989.

[17]M.S. Chen, K.G. Shin, "Adaptive Fault-Tolerant Routing in Hypercube Multicomputers",

IEEE Transactions on Computers, pp. 1406-1416, December 1990, Volume 39, No. 12.

[18]T.H. Cormen, C.E. Leiserson, R.L. Rivest,"Introduction to Algorithms", McGraw-Hill, MIT Press, 1990.

[19]W.J. Dally, C.L. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks", IEEE Transactions on Computers, Vol C-36, No. 5, pp. 547-553, May 1987.

[20]S.P. Dandamudi, D.L. Eager, "Hot-Spot Contention in Binary Hypercube Networks",IEEE Transaction on Computers, Vol. 41, No. 2, February 1992

[21]J. Duato, "A New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks", IEEE Transactions on Parallel and Distributed Systems, Vol. 4, No. 12, December 1993, pp. 1320-1331.

[22]S. Elliot, *Living with the Microchip*, The Bookwright Press, 1985

[23]S. A. Felperin, L. Gravanmo, G. D. Pifarre, and J.L.C. Sanz, "Routing Techniques for Massively Parallel Communications", Proceeding of the IEEE, Vol 79, No. 4, pp. 488-503, April 1991.

[24]W. K. Fuchs, E.E. Swartzlander Jr., "Wafer-Scale Integration: Architecures and Algorithms", Computer, April 1992, pp. 6-8.

[25]H. Fujiwara, "Logic Testing and Design for Testability", MIT Press, 1985

[26]J. Galiay, Y. Crouzet, M. Vergniault, "Physical vs. Logical Fault Models in MOS LSI Circuits: Impact on their Testability", IEEE Transactions on Computers, Vol C29, No. 6, pp.527-531, 1980.

[27]A. Gibbons, W. Rytter, "Efficient Parallel Algorithms". Cambridge University Press, 1988.

[28]Greene, J.W. and A. El-Gamel, "Configuration of VLSI Arrays in the Presence of Defects", J.ACM, Vol. 31, 1984, pp.694-717.

[29]C.J. Glass, L.M. Ni, "The Turn Model for Adaptive Routing", Technical Report MSU-CPS-ACS-44 Department of Computer Science, Michigan State University, October 1991

[30]C.J. Glass, L.M. Ni, "Fault-Tolerant Wormhole Routing in Meshes", Technical Report MSU-CPS-ACS-72 Department of Computer Science, Michigan State University, October 1992

[31]C.J. Glass, L.M. Ni, "Maximally Fully Adaptive Routing in 2D Meshes", Technical Report MSU-CPS-ACS-51 Department of Computer Science, Michigan State University, January 1992

[32]G. Grimmett, "Percolation", Springer-Verlag, 1980.

[33]D. G. Haenschke, D.A. Kettler, E. Oberer, "Network Management and Congestion in U.S. Telecommunications Network", IEEE Transactions on Communications, Vol. COM-29, No. 4, April 1981, pp. 376-385.

[34]M.C. Herbordt, C.C. Weems, J.C. Corbett, "Message-Passing Algorithms for a SIMD Torus with Coteries", ACM, pp. 69-78, 1990.

[35]R.W. Horst, "Task-Flow Architecture for WSI Parallel Processing", Computer, Volume 25,

Number 4, April 1992.

[36]IEEE Standard Test Access and Boundary Scan Architecture, IEEE Std. 1149.1-1990, IEEE Press, New York, 1990.

[37]U. Jagau, K.P. Dyck, H. Grabinski, "Power Distribution Strategies Based on Current Estimation and Simulation of Lossy Transmission Lines in Conjunction with Power Isolation Circuits", *Proceeding of the Internation Conference on Wafer Scale Integration*, pp. 288-297, San Francisco USA, 1990

[38]C. Jesshope, W. Moore, "Wafer Scale Integration", Adam Hilger, Bristol, England, 1986

[39]K.K. Johnstone, J.B. Butcher, "Power Distribution in Highly Parallel WSI Architectures", Proceeding of the Internation Conference on Wafer Scale Integration, pp. 203-214, 1989

[40]S. Kirkpatrick, C.D. Gelatt, M.P. Veechi, "Optimization by simulated annealing", Science 220, pp. 671-680, 1983.

[41]M. Kishinevsky, A. Kondratyev, A. Taubin, V. Varshavsky, "Concurrent Hardware", Wiley, 1994.

[42]T. Kohonen, "The Self Organizing Map", Proceedings of the IEEE, Vol. 78, No. 9, pp.1464-1480, Sept 1990.

[43]T. Kohonen, *Associative memory : a system-theoretical approach*, Springer-Verlag, 1977.

[44]T. Kohonen, *Content-addressable memories,* Springer-Verlag, 1980.

[45]T. Kohonen, *Content-addressable memories,* Springer-Verlag, 1987.

[46]T. Kohonen, *Self-organization and associative memory*, Springer-Verlag, 1984.

[47]S. Konstantinidou, L. Snyder, "The Chaos Router: A Practical Application of Randomization in Network Routing", 2nd Ann. ACM SAAP, pp.21-30,1990.

[48]Y. Lan, "Multicast in Faulty Hypercubes", 1992 Internation Conference on Parallel Processing", I-58-I-61.

[49]L. Lavagno, A. Sangiovanni-Vincentelli, *Algorithms for Synthesis and Testing of Asynchronous Circuits*, Kluwer Academic Publishers, 1993.

[50]B.H. Lavenda, "Brownian Motion", Scientific American, Volume 252, No. 2, February 1985.

[51]T. Leighton. C.E. Leiserson, "A Survey of Algorithms for Integrating Wafer-Scale Systolic Arrays", Wafer-Scale Integration, Elsevier Science, 1986, pp. 177-195.

[52]T. Leighton. C.E. Leiserson, "Wafer-Scale Integration of Systolic Arrays", IEEE Transactions on Computers", Vol. C-34, No. 5, May 1985, pp. 448-461.

[53]F.T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays Trees Hypercubes*, Morgan Kaufmann, 1992.

[54]C.E. Leiserson, B.M. Maggs, "Communication-efficient Parallel Algorithms for Distributed Random-Access Machines", Algorithmica, pp. 53-57, 1988.

[55]X. Lin, P.K. McKinley, L.M. Ni, "The Message Flow Model for Routing in Wormhole-Routed

networks", Technical Report MSU-CPS-ACS-78 Department of Computer Science, Michigan State University, January 1993

[56]X. Lin, P.K. McKinley, L.M. Ni, "Deadlock-Free Multicast Wormhole Routing in 2-D Mesh Multicomputers", IEEE Transaction on Parallel and Distributed Systems, Vol. 5., No. 8, August 1994, pp. 793-804.

[57]D.H. Linder, J.C. Harden, "An Adaptive and Fault Tolerant Wormhole Routing Strategy for k-ary n-cubes", IEEE Transactions on Computers, pp. 2- 12, January 1991, Volume 40, No. 1.

[58]J.L. McClelland, D.E. Rumelhart, *Explorations in Parallel Distributed Processing*, MIT Press, 1988.

[59]J.F. McDonald, E.H. Rogers, "The Trails of wafer-scale integration", IEEE Spectrum, pp. 32-39, Oct 1984.

[60]J.F. McDonald, R. Philhower, H.J. Greub, "Fine Grained, Highly Fault Tolerant Systems Based on WSI and FPGA technology", 2nd internation workshop on FPGAs.

[61]R.D. McLeod, J.J. Schellenberg, P.D. Hortensius, "Percolation and Anomalous Transport as Tools in Analyzing Parallel Processing Interconnection Networks", *Journal of Parallel and Distributed Computing*, Vol. pp. 376-387, January 1990.

[62]R. Negrini, M.G. Sami,R. Stefanelli, *Fault Tolerance Through Reconfiguration in VLSI and WSI Arrays,* MIT Press, 1989.

[63]L.M. Ni, P.K. McKinley, "A survey of Routing Techiniques in Wormhole Networks", Technical Report MSU-CPS-ACS-46 Department of Computer Science, Michigan State University, October 1991

[64]J.Y. Ngai, A Framework for Adaptive Routing in Multicomputer Networks, Technical Report Caltect-CS-TR-89-09, Computer Science Department, California Institute of technology, 1989.

[65]J.Y. Ngai, C.L. Seitz, "A Framework for Adaptive Routing in Multicomputer Networks",Computer Architecure News, Vol. 19, No. 1,pp. 6-14, March 1991.

[66]R.V. Pelletier, *Alternative Approaches in Improving Fault Tolerance for a Wafer Scale Environment*, M.Sc. Thesis, University of Manitoba, 1993.

[67]R.V. Pelletier, D.C. Blight, R.D. McLeod, "Fault Tolerance in a Wafer Scale Environment", 1993 International Conference on Wafer Scale Integration, pp.173-184, San Francisco USA, 1993.

[68]G.D. Pifarre, L.Gravano, S. A. Felperin, J.L.C. Sanz, "Fully Adaptive Minimal Deadlock-Free Packet Routing in Hypercubes, Meshes, and Other Networks: Algorithms and Simulations", IEEE Transactions on Parallel and Distributed Systems, Vol. 5, No. 3, March 1994, pp. 247-263.

[69]D.K. Pradham, *Fault Tolerant Computing Theory and Techniques Volume 1*, Prentice Hall, 1986.

[70]D.K. Pradham, *Fault Tolerant Computing Theory and Techniques Volume 2*, Prentice Hall, 1986.

[71]Rosenberg, A.,"The Diogenes Approach to Testable Fault-Tolerant Networks of Processors, IEEE Trans. on Computers, C-32, 1983, pp.902-910.

[72]Stanley H.E., *Form: An Introduction to Self-Similarity and Fractal Behavior, On Growth and Form*, Ed. H.E. Stanley and N. Ostrowsky, Martinus-Nijhoff Pub. Co., Boston, 1986, pp.21-53.

[73]J.J. Schellenberg, *Percolation Theory For Wafer-Scale Arrays,* PhD. Thesis, University of Manitoba, 1990.

[74]Scher, H. and E.W. Montroll. "Anomalous transit-time dispersion in amorphous solids", Phys. Rev., B12, 1975, pp.2455-2477.

[75]C.L. Seitz, "Let's Route Packets Instead of Wires", Proc. of the Sixth MIT Conference : Advanced Research in VLSI, pp. 133-138.

[76]A.D. Singh, "Interstitial Redundancy : An Area Efficient Fault Tolerance Scheme for Large Area VLSI Processor Arrays", IEEE Transactions on Computers, Vol 37, pp. 1398-1410, 1988.

[77]D. Staffer, "Introduction to Percolation Theory", Taylor and Francis, 1985.

[78]J. Staunstrup, "A Formal Approach to Hardware Design", Kluwer Academic Publishers, 1994.

[79]L. Valiant, G. Brebner, "Universal Schemes for Parallel Communication", ACM STOC, pp.263-277, 1981.

[80]T.W. Williams, K.P. Parker, "Design for testability - A survey", IEEE Transactions on Computers, Vol. C-31, No. 1, Jan. 1982, pp.2-15.

[81]T.W. Williams, *VLSI Testing*, North-Holland, 1986.

[82]Xilinx, *The Programmable Gate Array Data Book*, Xilinx 1991.

Appendix A: Kohonen Maps For Spatial Organization

In addition to the problem of routing messages in a faulty environment, difficulties in other aspects of utilizing a WSI environment for parallel processing are also of interest. One such problem is process allocation, the determination of a mapping of parallel processes to actual processing elements. It is important to obtain such a mapping which conforms to the irregular topology on the network in order to minimize network traffic and optimize performance. The Kohonen Self Organizing map is an unsupervised neural network learning algorithm developed for speech recognition problems. It is hoped that such non traditional techniques can be used in studies of problems in WSI.

In this appendix, the Kohonen self-organizing map is briefly discussed, and an application of it to a VLSI problem is presented. A discussion of parallel implementations of the algorithm specifically on mesh topologies is presented.

A.1 Kohonen Self Organizing Map

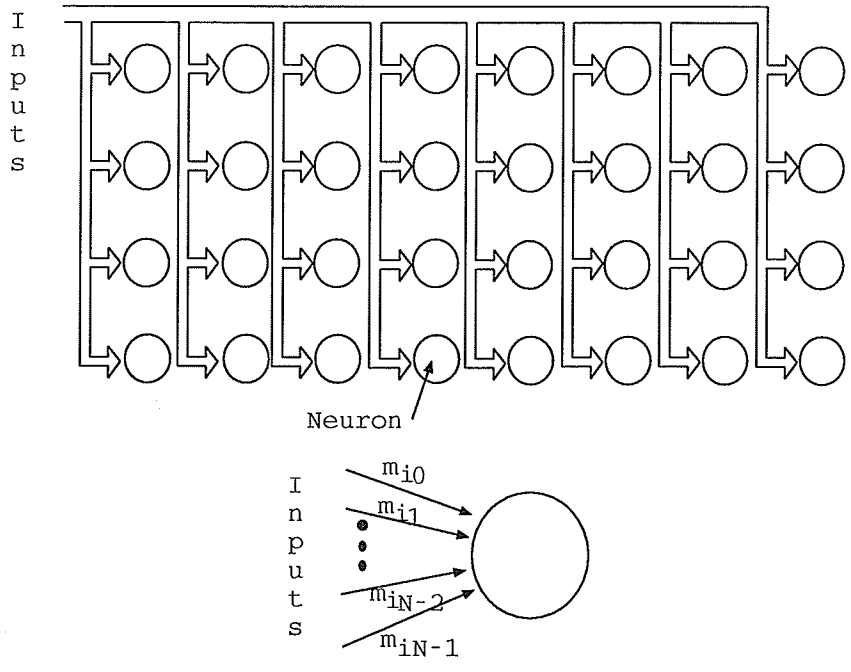


Figure 67 Kohonen Network

The Kohonen self organizing map is an unsupervised learning algorithm for ANN initially developed for applications including speech and pattern recognition[90], and content addressable memories[86][87][88][89]. Although the FPGA placement problem does not appear to be closely related to the pattern recognition problem, the self organizing map produces a spatially ordered map of its input signals. It is this spatial ordering which we will utilize in the placement algorithm.

The Kohonen map exhibits two essential effects which produce the spatial ordering[85].

- spatial concentration of neural activity in a neighbourhood which best matches the input. This is a product of the competitive nature of the learning algorithm.
- Weight adjustment of the neuron and its topological neighbourhood whose weights most closely match the input vector and its topological neighbourhood.

In this section we will present a brief overview of the Kohonen self-organizing map, including the basic neural model for these networks, and the unsupervised learning algorithm.

A.2 Neural Model

The basic structure of the neural network used in Kohonen maps is show in Figure 67. A two dimensional array of neurons are laid out in a rectangular fashion. Each neuron is connected to the N input signals of the network. The input signals can be represented as an N-dimensional vector.

$$x = [x_0, x_1, \ldots, x_{n-1}] \in \Re^N$$

Each neuron in the network is connected to each of the input signals, and each connection has a weight associated with it. The weights for each neuron i can also be represented as a vector.

$$m_i = [m_{i0}, m_{i1}, \ldots, m_{in-1}] \in \Re^N$$

We shall restrict the range of input and weight values to the interval [0,1]. This will not affect the results presented here.

A.3 Learning Algorithm

Learning refers to the determination of weight values on connections in a neural network. Although it may be possible to analytically determine the optimal weights for a network, it is usually more convenient to employ learning algorithms which allow networks to determine their weights based on training sets of input/output signals. Unsupervised learning differs from the supervised learning algorithms traditionally employed in ANN in that no output values are specified in the training set.

The learning of weights proceeds by repeatedly selecting an input vector from the training set, and finding the best match between neuron weight vectors and the input vector (training vector labelled x). The best match refers to the neuron (j) in the network such that:

$$|x - m_j| = min\ (|x - m_i|)$$

where the Euclidean norm is typically used to measure distance.

Once the best match is found, the weights of the connections to the neuron and those of its neighbours are adjusted according to the following:

$$m_i(t+1) = m_i(t) + h_{ji}(t)\ [x(t) - m_i(t)]$$

$$h_{ji}(t) = h_0(t)\ e^{-|r_i - r_j|/\sigma(t)2}$$

where $|r_i - r_j|$ is a measure of the distance between neuron i and j (the best match). $h_0(t)$ and $\sigma(t)$ are functions decreasing with time.

A.4 Kohonen Map Placement Algorithm

The popularity of Field Programmable Gate Arrays (FPGA) has been dramatically increasing in recent years. One family of these devices consisting of a large array of identical programmable Lookup-Tables (LUTs) blocks connected by programmable interconnects enables these devices to be programmed to implement both combinational and sequential circuits. Although each FPGA device may contain a large number of LUTs, it is not always possible to utilize all of the available resources for a particular design. In order to increase the utilization of available resources it is important to develop tools which can efficiently synthesize design specifications into a programmed device. FPGA tools typically divide this process into three steps. First the design specification is partitioned into simple functions, each of which is implementable by the LUTs in the device. Next each function is assigned to a specific LUT in the FPGA device. Finally the interconnects between functions are routed. In this chapter we focus on the placement problem of assigning functional tasks to individual LUTs in the FPGA such that constraints based on area and performance are optimized.

Placement algorithms utilized in CAD tools for FPGAs have traditionally employed algorithms originally developed for other technologies including Printed Circuit Boards (PCB) and Integrated Circuit (IC) layouts. Although the placement problem for FPGA is closely related to those of other technologies, the basic structure of FPGA is different from other technologies. FPGA are implemented as a prefabricated array of programmable logic blocks and interconnects. The regular structure of such devices make the use of alternative placement algorithms attractive. In this chapter we present a placement algorithm developed for FPGAs based on the unsupervised learning algorithms used in Artificial Neural Networks (ANN) [90]. The Kohonen self organizing map[85] is used to first map the connectivity of the design to a two dimensional regular mesh topology, and then simple one and two dimensional compaction algorithms may be used to produce an area efficient and highly routable mapping.

The Kohonen learning algorithm discussed in the previous section has the special property of producing spatially organized representations of the input training set. This process may also be

thought of as dimensional reduction. The higher dimensional input vector set is reduced to a representation in 2 dimensions (the dimension of the neuron array). In this section we discuss the application of the Kohonen learning algorithm to the placement problem for FPGAs. The basic approach is to have the training data model the connectivity of the circuit to be placed. The corresponding map produced by the Kohonen learning algorithm may then be used to determine the LUT to which functions are to be assigned.

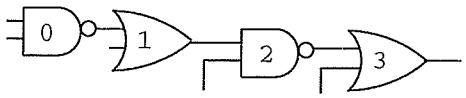
A.5 Circuit Representation

Let M be the number of gates in the circuit to be placed. A gate refers to a function which is to be implemented by a LUT. The circuit C may be represented as a graph G,

$$G = (V,E)$$

$$V = (v_0, v_1, \ldots, v_{N-1})$$

where V, the vertices of the graph, is the set of gates in the circuit. E is the set of edges in the graph. An edge exists between two vertices if there is a direct connection between the corresponding gates. An example is shown in Figure 68.



$X_0 = (1, 1/2, 1/3, 1/4, \ldots)$
$X_1 = (1/2, 1, 1/2, 1/3, \ldots)$
$X_2 = (1/3, 1/2, 1, 1/2, \ldots)$
$X_3 = (1/4, 1/3, 1/2, 1, \ldots)$

Figure 68 Circuit Representation

The set of training vectors T, is a set of n-dimensional vectors:

$$T = (x_0, x_1, \ldots, x_{M-1})$$

$$x_i = (x_{i0}, x_{i1}, \ldots, x_{iN-1})$$

such that

$$x_{ii} = 1.0$$

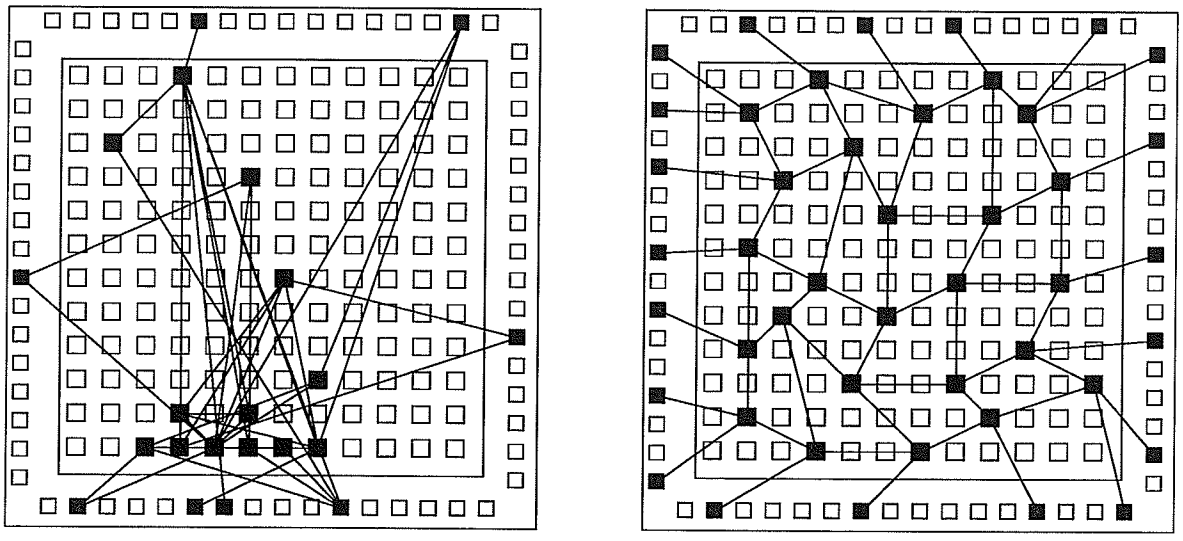
$$x_{ij} = \setminus 1/d(v_i,v_j) + 1 \qquad for\ i \neq j$$

where $d(v_i,v_j)$ is the distance between vertices v_i and v_j.

A.6 Basic Learning Algorithm

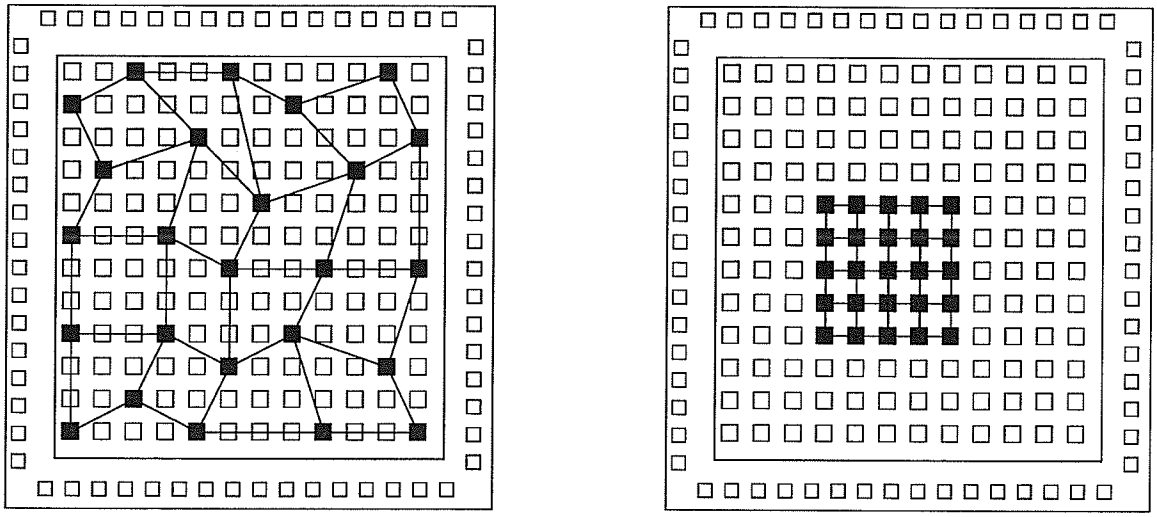
The basic Kohonen learning algorithm described in the previous section is usable with minor modifications. The basic layout of a Xilinx type FPGA[92] is shown in Figure 69. The device contains a square array of Combinational Logic Blocks (CLBs) surrounded by a ring of I/O Blocks (IOBs). Each block in the FPGA device corresponds to a neuron in the Kohonen network.

In order to prevent gates requiring I/O blocks from being assigned to CLBs, and gates implementing CLB function from being assigned to IOBs, the best match criteria is modified to distinguish between IOBs and CLBs. If a training vector corresponds to a gate requiring a IOB, only the IOBs in the device are searched for the best available match. Likewise, only CLBs are searched for the best match for training vectors corresponding to CLBs. The distinction does not affect the selection of neighbourhoods around blocks.



(a) (b)

Figure 69 Kohonen Placement



(a) (b)

Figure 70 Kohonen Placement

An example of a placement of a simple regular circuit structure is shown in Figure 69. The circuit being placed is a simple 5 by 5 mesh of gates, with all edge gates connected to outputs. This example was chosen because it allows the behaviour of the algorithm to be clearly seen. Each black square in the figure indicates that a gate has been assigned to that block. This means that the vector associated with one of the gates has that block as its best match. It can be seen that the layout topology matches the original topology.

A.7 Extended Circuit Representation

Although the layout in Figure 69 looks nice, it is not the optimal layout in terms of any measure of placement quality. It is important to realize that the goal of the Kohonen algorithm is not specifically to optimize the placement according to some cost function, but instead to find a topological map which corresponds to the training set. In order to produce a placement which minimizes the area and wire cost, some modifications to the basic algorithm are required.

One of the first modification we explored was to change the circuit representation for circuit so that the dimensionality of the training set N is equal to the number of blocks in the FPGA device, and not the number of gates in the circuit. The additional training vectors contain all zeros except for a value of 1.0 in the ith position. This process is equivalent to adding dummy gates to the netlist which are not connected to any other gates. These dummy gates would be removed at

the end of the placement.

A.8 Two Phase Placement

Although the modification to the algorithm in the previous section does produce a better placement than the original algorithm, it does suffer from a few practical problems. First of all it dramatically degrades the performance of the algorithm by increasing the dimensionality of the problem. Secondly, the basic algorithm does not offer any easy method to introduce external constraints in the placement such as routing priorities.

An alternative approach is to separate the placement procedure into two phases. In the first phase, the basic Kohonen placement algorithm is used. In the second phase traditional one or two dimensional compaction algorithms are used. An example using simulated annealing with a temperature of zero for the second phase is shown in Figure 70. Alternative compaction algorithms would be equally or better suited.

A.9 Results

In this section we will present some comparisons of the two phase Kohonen placement algorithm. We will compare the algorithm to the simulated annealing algorithm. Our measure of performance will be based on the time to perform placement, and a measure of the total distance of wire required to connect the circuit.

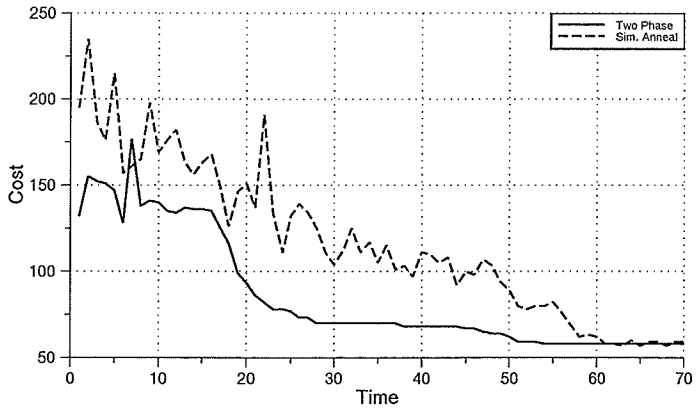


Figure 71 Comparison of Algorithms

Figure 71 shows a comparison of the cost of the placement versus time for both the two phase Kohonen placement (solid line) and a simulated annealing placement (dashed line). The two phase placement uses the Kohonen placement for the first 16 time units, and then uses a 0 temperature simulated annealing placement for the remainder of the placement. It can be seen the Ko-

honen phase of the algorithm does not attempt any minimization of the cost function, however, once this phase is complete, rapid compaction is achievable.

The following is a comparison of the two algorithms for three different circuits.

Circuit	Kohonen		Sim Anneal	
	Time	Cost	Time	Cost
Mesh	100	52	250	52
Mesh*	110	68	250	65
ALU	250	120	300	115

Mesh* refers to a mesh with wrap around in the horizontal direction.

A.10 Parallel Implementation of Kohonen Maps

It is very straight forward to implement the Kohonen algorithm on a serial processor. The state of each neuron is stored in a large array and the algorithm proceeds by processing each neuron's activity sequentially one after another. It can easily be seen that the algorithm has a time complexity for each iteration of $\vartheta(N)$ where N is the number of neurons. The Kohonen algorithm is likely an NP-complete problem due primarily to its similar nature to simulated annealing[83].

In this section I will briefly consider parallel implementations of Kohonen's learning algorithm. I will first discuss an idealized implementation and then discuss some practical implementation of parallel versions. The actual implementation of the algorithm on a transputer configured as a mesh will as be presented.

A.11 Parallel Hardware for Kohonen's Algorithm

A constant time complexity per iteration parallel implementation is possible for Kohonen's algorithm, however it requires hardware which may not be practical for actual implementation. One such possible implementation is shown in Figure 72. In this figure:

- X is the input vector selected from the training set.
- h_0 a parameter in the learning algorithm.
- σ is a parameter in the learning algorithm.
- ϕ is a clock signal used to control when the minimum output is available.
- $\|X\text{-}M\|$ is the distance between input vector X and weight vector M.

- C is the coordinate of a neuron.

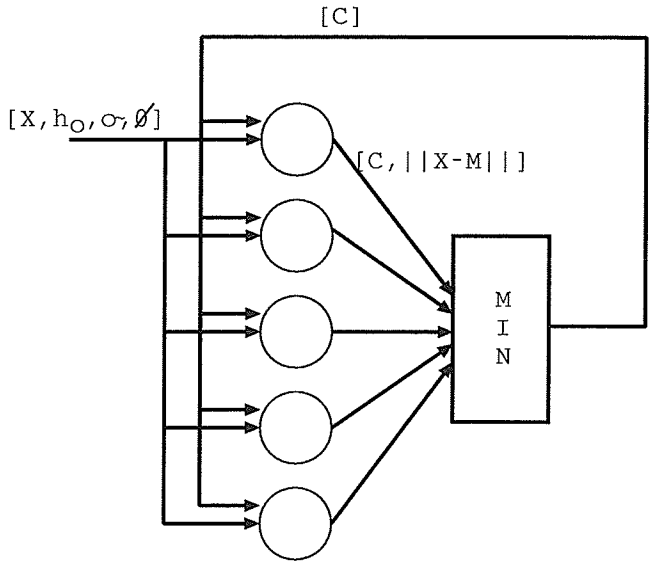


Figure 72 Neural Implementation

The network operates by passing the training vector X, simulation parameters h_0 and σ to each neuron processor. Each neuron compares this vector to its current weight vector and transmits the distance between them and the coordinate of the neuron to the minimum circuit. The minimum circuit calculates the minimum of all the $\|X\text{-}M\|$ and outputs the coordinate of the best match back to all the neurons. Each neuron them updates its weights according to equation (5.5). This implementation requires a constant time minimum function which can be implemented using a CRCW PRAM [84], as well a unlimited fan-out for both the minimum circuit and input driver. Such an implementation would not be practical for implementation of large networks.

A.12 Practical Parallel Implementations

The primary difficulty with the implementation discussed in the previous section was that we were using a constant time minimum function. The construction of such a circuit is not practical given current technology for large N. We would also like to construct a circuit out of identical processing elements. In this section we examine the use of tree structure of processors for use with the Ko-

honen learning algorithm.

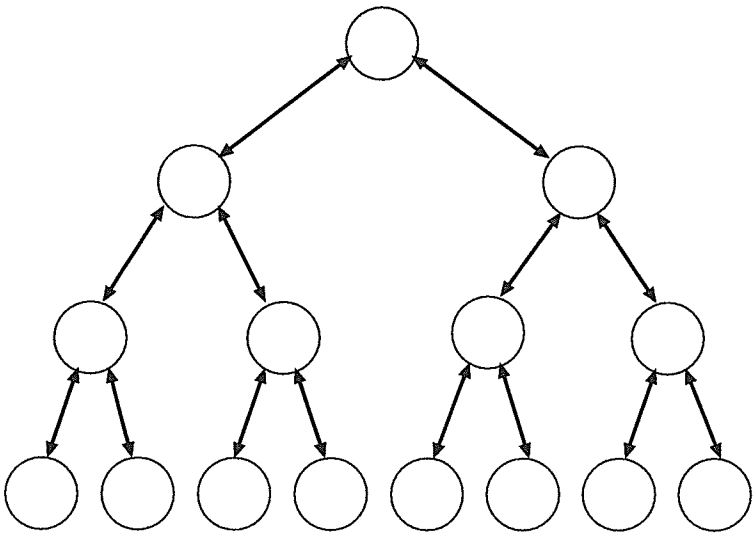


Figure 73 Tree Architecture

In Figure 73 a binary tree configuration is shown. Each of the nodes in the network is an identical processor. We wish to implement a P neuron network on this network with N leaf nodes, and we assume $P \geq N$. We assume that only leaf nodes simulate neuron activity.

Simulation of each iteration of the Kohonen learning algorithm proceeds as follows:

1. The root node is passed a vector from the training set and the current simulation parameters h_0 and σ.
2. Each branch node (non-leaf node) passes on the input vector and simulation parameters from its parent to its children connections.
3. When a leaf node receives a training vector, it compares the vector to the weight vector of each of neurons it is simulating.
4. The coordinate of the best match (minimum $\|X\text{-}M\|$) and the distance from step 3 is sent to the parent connection of each leaf node.
5. Each branch node receives two closest matches from its child connections, and passes on the better match of these two to its parent.
6. The root node contains the best match of all neurons, and the coordinate of the neuron. This value is passed down to all the leaf-nodes in the network.
7. Each leaf node updates all its neuron weights according to equations (5.4) and (5.5).

It is not hard to verify that this is $\vartheta(log(N))$ algorithm. It is possible to improve the efficiency of this implementation by utilizing branch nodes for neuron simulation, and using a higher degree tree structure. These changes will only offer a constant factor improvement in performance.

A.13 Mesh Implementation

Although the tree structure discussed in the previous section offers a reasonable implementation, for large networks a mesh connected topology offers advantages of only nearest neighbour connections, and constant wire lengths for interconnections. In this section I will discuss the implementation of a Kohonen simulator on a transputer configured in a mesh topology.

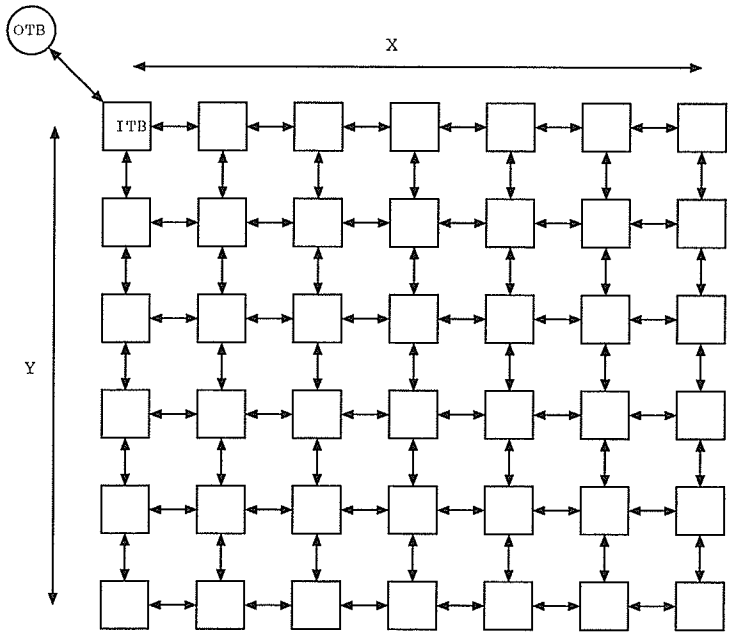


Figure 74 Mesh Architecture for Parallel Implementation

The implementation discussed in this report was implemented on a 16 node transputer, using the Trollius software[91]. The transputer was configured into a mesh topology shown in Figure 74, although not all the connections in the mesh will be utilized.

The Trollius software allows one node in the network to be run on the host computer. This node is identified in Figure 74 by the round node and labelled OTB (for Out of The Box). This node performs all the I/O operations and user interface operations for the program. All mesh nodes (referred to as ITB for in the box), perform the actual simulation. A two dimensional mesh of processors is used for all neuron simulations.

The algorithm proceeds as follows:

1. The OTB node reads all user inputs including training set and simulation parameters such as h_0 and σ.
2. Each node in the network initializes the state of the *P/N* neurons it is simulating.
3. The OTB node repeats the following loop until the simulation is complete:

4. A vector from the training set is selected at random. This value is passed to the (0,0) node in the mesh.
5. The OTB then waits until a message is received from node (0,0). The message will contain the coordinate of the neuron with the closest match to the training vector.
6. A message is sent to node (0,0) with the coordinate of the best match, and appropriate simulation values (h_0 and σ).
7. simulation time variables and parameters are updated.
8. Process repeats.
9. Each ITB waits until a message is received from one of its neighbours. The action performed depends upon the type of packet received:

- COMPARE The packet contains a test vector from the training set. The packet is passed to the node to the right of the current node, and to the node below the current node if the current node is on the left edge of the mesh. The current node then compares the vector to the weights of all neuron that it is simulating. If the node is on the right edge of the mesh, a MATCH packet containing the closest match is sent to its left neighbour.
- MATCH The packet contains the closest match found so far. If the match is better than that of the current node, the best match value of the current node is update to that of the packet. If the current node not a left edge node, the best match is passed to the left neighbour of the current node. If the current node is on the left edge of the mesh and MATCH packets have been received from both its right and bottom neighbours, the best match is passed to the current nodes top neighbour. If the current node is (0,0), the best match is passed to the OTB.
- UPDATE The packets contains the coordinate of the best match in the network and current simulation parameters. The packet is passed to the node to the right of the current node, and to the node below the current node if the current node is on the left edge of the mesh. All neurons being simulated by the current node have their weight adjusted according to equations (5.4) and (5.5).
- TERMINATE The packet tells the process on the node to terminate after forwarding the packet to its right neighbour and to its bottom neighbour (if node is one left edge of mesh).

Because each iteration of the algorithm requires a packet to be sent across the network three times, each traversal requires $2\sqrt{N}$-1 steps the algorithm is $\vartheta(\sqrt{N})$.

No performance measures are given due to the difficulty in measuring time in a multiuser environment and the limited number of processors available. Qualitative analysis of running the program on different size meshes indicates that the algorithm is approximately $\vartheta(logN)$.

A.14 References

[83]E. Aarts, J. Korst, "Simulated Annealing and Boltzmann Machines", Wiley, 1989.

[84]A. Gibbons, W. Rytter, "Efficient Parallel Algorithms". Cambridge University Press, 1988.

[85]T. Kohonen, "The Self Organizing Map", Proceedings of the IEEE, Vol. 78, No. 9, pp.1464-1480, Sept. 1990.

[86]T. Kohonen, "Associative memory: a system-theoretical approach", Springer-Verlag, 1977.

[87]T. Kohonen, "Content-addressable memories", Springer-Verlag, 1980.

[88]T. Kohonen, "Content-addressable memories", Springer-Verlag, 1987.

[89]T. Kohonen, "Self-organization and associative memory", Springer-Verlag, 1984.

[90]J.L. McClelland, D.E. Rumelhart, Explorations in Parallel Distributed Processing, MIT Press, 1988.

[91] G. D. Burns,``A Local Area Multicomputer",Proceeding of the Fourth Conference on Hypercubes, Concurrent Computers, and Applications, 1989.

[92]Xilinx, The Programmable Gate Array Data Book, Xilinx 1991.*

Appendix B: Routing for MINs

B.1 Introduction

Many large systems including parallel processing systems and telecommunication networks often have a need to provide multiple simultaneous connectivity between a large number of input and output terminals. As the number of terminals increase, it is usually not possible or necessary to implement full connectivity between all terminals, as the number of connections and cost grows exponentially. Typically such systems will utilize a interconnection methodology which will provide multiple simultaneous connections between all terminals, but can not connection all input output permutations simultaneously. Multistage Interconnect Networks (MINs) are one means of implementing these networks providing connectivity between input and output terminals utilizing multiple stages of simple switching elements.

The shuffle-exchange or Omega network is a commonly used network topology for computer networks and interconnection networks. One of the attractive features of such networks is the relatively easy routing between input and output terminals of the network [94][95]. Normally the size of such networks is restricted to the case where N (the number of input terminals) is a power of two. With this assumption, a path from an arbitrary input terminal can be found to an arbitrary output terminal. The ease in calculation is a result of the observation that each shuffle permutation can be represented as a circular left shift.

In most practical applications it is not cost effective to restrict the network size to be a power of two. If we wish to increase a relatively large network by a small number of terminals, we may be forced to double the size of the network, and have unused terminals at both inputs and outputs. Padmanabhan [93] introduced even-sized shuffle-exchange networks in which the network size simply has to be an even integer. It was shown that such networks possess relatively easy to calculate control tags through the network, which do not require any significant increase in the complexity of the routing algorithms. Performance in such networks was found to be comparable to that of networks with size restricted to a power of two. These networks also offer some performance and fault tolerant properties as the edge disjoint paths exists for many some of the input-output terminal pairs.

Fault tolerance is an important aspect of the design of MIN. cost and reliability are two important issues in the design os any system. In the design of large systems such as MINs, fabrication defects may be an important issue in the cost of such networks. All large systems will contain manufacturing defects. It is cost effective to attempt to design systems which work in the presence of faulty, instead of repairing defective systems. Reliability is also important in most applications, and systems where faults do not cause catastrophic effects (such as system failure) are highly desirable. For these reasons we wish to develop fault tolerant MINs.

In this appendix we further extend the result of Padmanabhan [93] by developing a bidirectional network and routing algorithms for fault tolerant routing in these networks. In such a net-

work it is possible to implement an adaptive fault tolerant routing algorithm which can bypass faulty switching elements in these networks. Padmanabhan was concerned with interconnect networks which were circuit switched. Fault tolerance was achieved by utilizing disjoint paths between input and output terminals which exists for some permutations. Although a degree of fault tolerance was obtained, this approach does not guarantee that a connection can be made even in the presence of a single fault. If an alternative path exists between input and output terminal they are disjoint. Our approach is to develop a routing algorithm which can utilize the connectivity of the network and guarantee delivery of a message through the network in the presence of a fault.

The remainder of this appendix is organized as follows. Section two discusses properties and configuration of even-sized shuffle-exchange networks and discusses the determination of control tags to connect output to inputs. The next section discusses some applications of the new control tags to fault tolerant routing in these networks. The final section summarizes results, and discusses future work.

B.2 Even-Sized Shuffle-Exchange Networks

In this section we discuss the even-sized shuffle-exchange network originally presented by Padmanabhan[93], and extend the network so that it can route bidirectionally. We will develop control tags which can route both forward and backwards in these networks. These control tags will be used in the fault tolerant routing algorithm presented in the following section.

B.2.1Shuffle-Exchange Networks

A N X N shuffle-exchange network is constructed by cascading layers of perfect shuffle connections and exchange elements (see Figure 1). A perfect shuffle on N' terminals (where N' is even) interleaves the first N'/2 terminals with the last N'/2 terminals in an analogous operation to shuffling playing cards. This operation may be represented by the permutation π on terminals 0,1,..., N'-1 as follows:

$$\pi(i) = \left(2i + \left\lfloor \frac{2i}{N'} \right\rfloor\right) \bmod N'$$

Let I

In most shuffle exchange networks N' is restricted to values which are powers of two. This simplifies the permutation operation so that it can be represented as a circular left shift. In this appendix we are concerned with the case where N' is not a power of two.

$$N' = N + M, \quad N = 2^n, \quad 0 < M \leq N$$

Although the ESSE network topology can be utilized for both direct and indirect networks, MINs are normally constructed using circuit switched

Although the ESSE network can be utilized for both direct and indirect networks, in the de-

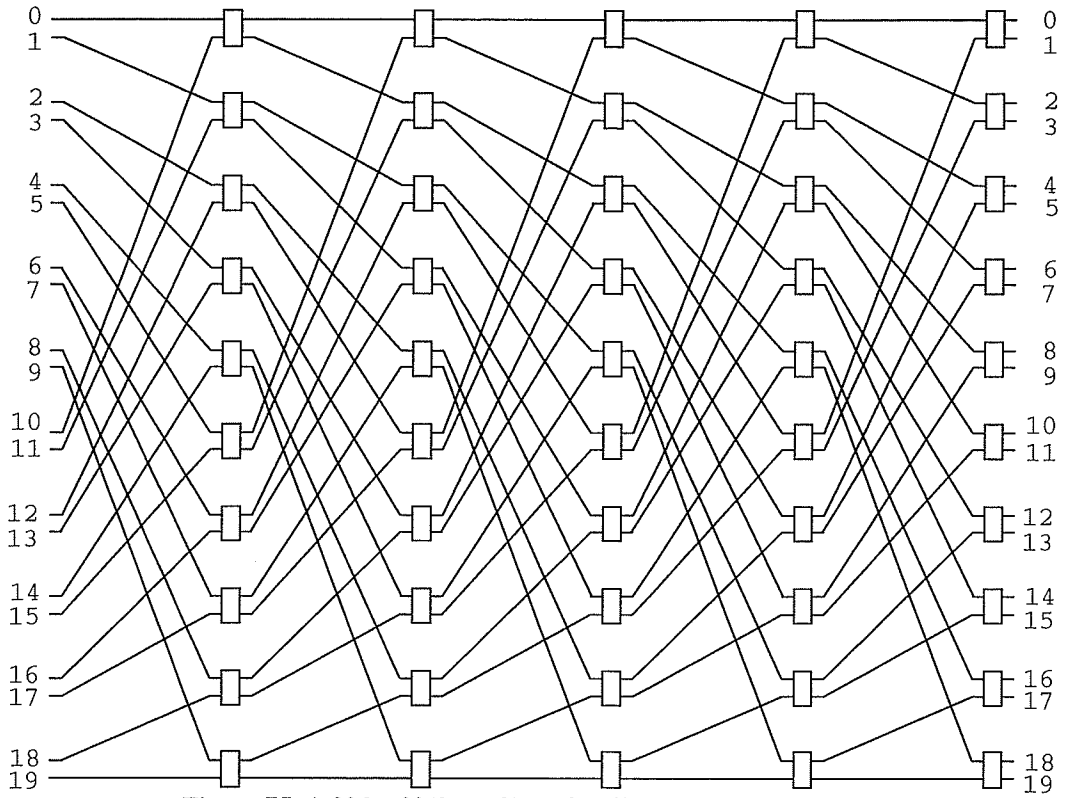


Figure 75 A 20 by 20 Even Sized Shuffle Exchange Network

sign of MINs we are concerned primarily with circuit switched networks. Each switching element in the networks, is a simple 2 X 2 switch which can assume any of the configurations shown in Figure 76.

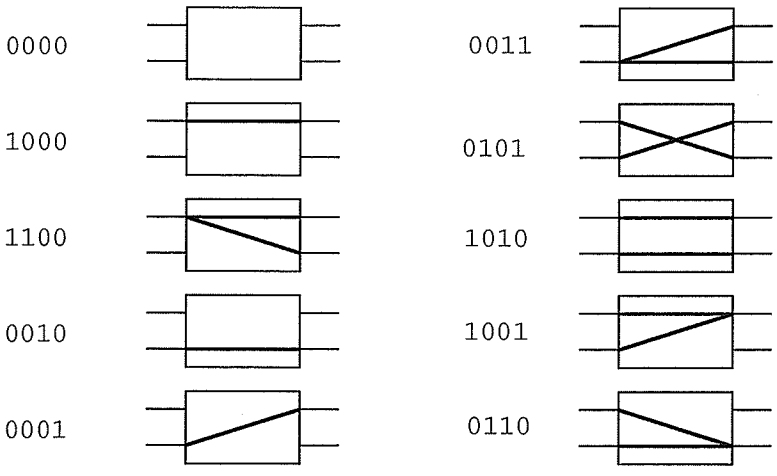


Figure 76 Circuit Switched Exchange Elements

Let an even-sized shuffle exchange network be represented by a tuple G:

$$G = (E,I,O)$$

where

E is the set of exchange elements

I is the set of input terminals

O is the set of output terminals

$$E = \{e_{rc} | 0 \le r < \frac{N'}{2}, 0 \le c < \lceil \log N' \rceil\}$$

e_{rc} is the exchange element at row r column c

$$I = \{i_r | 0 \le r < \dot{N}\}$$

$$O = \{o_r | (0 \le r < \dot{N})\}$$

We further label a line l_{rc} as the r^{th} input to the c^{th} shuffle exchange layer

A ESSE network is constructed of $\lceil \log N' \rceil$ consecutive stages of shuffle permutations and $\frac{N'}{2}$ switching elements.

B.2.2Forward Routing

Given a multistage shuffle-exchange network as described in the previous section we wish to be able configure a path from an arbitrary input to an arbitrary output terminal (referred to as *i* and *j* respectively). To accomplish this we need to utilize a control tag which configures the exchange elements to create this path. Let a control tag *T'* be represented as follows[93]:

$$T' = 2^n t'n + 2^{n-1} t'_{n-1} + ... + 2t'_1 + t'_0$$

where t'_k is used to control the appropriate exchange element in the $\lceil N' \rceil - k$ column. Alternatively this can be viewed as the left most bit controls the left most exchange element along the path. Each successive bit corresponds to each successive exchange elements along the path.

Padmanabhan [93] shows that control tags may be determined using the following equations:

$$T'_1 = (j + 2Mi) \mod N'$$

$$T'_2 = T'_1 + N', \qquad \text{iff} \quad T'_1 + N' < 2N$$

It was also shown by Padmanabhan [93] that if two control tags exist, that the paths used by these tags are disjoint. Further more all but 2M input-output pairs have two paths between them.

The above equations allow us to route from any input terminal to any output terminal. It may be necessary sometimes to route from internal nodes in the network to other internal nodes in a later column (this will be required in the next section for fault tolerant routing). To accomplish this we utilize the following theorem

Theorem 21

A control tag T' which will route from line $i = l_{r,c}$ to line $j = l_{r',c+k}$,k columns to the right of i ($0<k<\lceil \log N' \rceil$) can be calculated as follows:

$$T' = r' - r2^k + \left\lfloor \frac{r2^k}{N'} \right\rfloor N' \qquad r' >= 2^k\text{-}M$$

$$T' = r' - r2^k + \left\lfloor \frac{(r+1)2^k}{N'} \right\rfloor N' \qquad r' < 2^k\text{-}M$$

Proof of Theorem 21

$$r' = (\pi^k(r) + T'') \mod N'$$

$$r' = \left(r2^k + \left\lfloor \frac{r2^k}{N} \right\rfloor + T''\right) \mod N'$$

$$r' + mN' = r2^k + \left\lfloor \frac{r2^k}{N'} \right\rfloor + T''$$

$$\text{where } m = \left\lfloor \frac{\left(r2^k + \left\lfloor \frac{r2^k}{N'} \right\rfloor + T''\right)}{N'} \right\rfloor$$

$$T' = r' - r2^k - \left\lfloor \frac{r2^k}{N'} \right\rfloor + mN'$$

$$T'' = r' - r2^k + \left\lfloor \frac{r2^k}{N'} \right\rfloor N' \text{ iff } r' >= 2^k\text{-}M$$

else

$$T'' = r' - r2^k + \left\lfloor \frac{(r+1)2^k}{N'} \right\rfloor N' \qquad r' < 2^k\text{-}M$$

B.2.3Reverse Routing

Consider a control tag T' used to route a packet from terminal i to terminal j. If we wish to return a packet from terminal j to terminal i, we can not use control tag T'. This can be seen in example 1 in appendix A where in case 1 and 2 the same control tag (T' = 0) routes a packet from terminals 1 and 16 (See appendix A). Clearly this tag can not be used to return a message. Because of this characteristic (we say this not is not a Delta network), calculation of control tags in more difficult in both the forward and reverse directions.In this section we will show how to calculate a reverse control tag, which can route from output terminals to input terminals, and also control tags which can route partially through a network in the reverse direction.

Given a multistage shuffle-exchange network identical to the one in the previous section we wish to be able configure a path from an arbitrary output terminal (j) to an arbitrary input terminal (i). Let a control tag *T''* be represented as follows[93]:

$$T'' = 2^n t''_n + 2^{n-1} t''_{n-1} + \ldots + 2t''_1 + t''_0$$

where t''_k is used to control the appropriate exchange element in the $\lceil N' \rceil - k$ column. Alternatively this can be viewed as the left most bit controls the left most exchange element along the path. Each successive bit corresponds to each successive exchange elements along the path.

Theorem 22

A control tag T'' which will route from terminal j ($l_{r'c+k}$) to terminal i(l_{rc}), k columns to the left of j (k > 0) can be calculated as follows:

$$T'' = \left\lfloor \frac{(r+1)2^k}{N'} \right\rfloor \text{ if } (N' + r') - (r2^k) \bmod N' < 2^k$$

$$T'' = \left\lfloor \frac{r2^k}{N'} \right\rfloor \text{ if } r' \geq (i2^k) \bmod N' \text{ and } r' < (r2^k) \bmod N' + 2^k$$

Proof of Theorem 22

Condition 1 : $(N' + r') - (r2^k) \bmod N' < 2^k$

Condition 2 : $r' \geq (i2^k) \bmod N'$ and $r' < (r2^k) \bmod N' + 2^k$

Theorem 23

A control Tag T'' which will route from output terminal j to input terminal i can be calculated as follows:

$$T''_1 = 2^i - \left\lfloor \frac{2Mi + j}{N'} \right\rfloor$$

and a second control tag T''''_2 exists if $(j + 2Mi) \bmod N' + N' < 2N$,

$T_2 = T_1 + 1$

Proof of Theorem 23

Proof of this theorem follows from Theorem 2, using $k = \lceil \log N' \rceil$

B.2.4Bidirectional Routing

Given a multistage shuffle-exchange network as described in the previous section we wish to be able configure a path from an arbitrary output terminal (j) to an arbitrary input terminal. To accomplish this we need to utilize a control tag which configures the exchange elements to create this path. Let a control tag *T'* be represented as follows[93]:

$$T' = 2^{2n+1}t''_n + 2^{2n}t'_n + \ldots + 2t''_0 + t'_0$$

where t'_k and t''_k are used to control the appropriate exchange element in the $N' \mid -k$ column. Alternatively this can be viewed as the left most bit controls the left most exchange element along the path. Each successive bit corresponds to each successive exchange elements along the path. We are assuming that we are utilizing the simple exchange elements shown in Figure 76b.

B.3 Fault Tolerant Routing

In this section we discuss some applications of the previous section to interconnection networks, with emphasis on fault tolerance. One of the most useful properties of the even sized networks is that for small values of M ($N' \approx N$) we have a large number of edge disjoint paths between input and output terminals. In a large network, in the presence of faults (in either the connections or exchange elements) it is important to have this sort of mechanism to allow routing of messages which can bypass faulty parts of network, and ensure reliable delivery of messages.

B.3.1Fault Model

In this appendix we will assume that only a single fault can occur in the network at one time.Based on this simple assumption we will develop routing algorithms which are capable of delivering faults in the presence of these faults. In this section we will describe the fault model we are using for this work.

We assume that faults can only occur in switching elements. This assumption is justified by two observations of interconnect networks. First that the complexity of the switching element will be much greater than that of the interconnect, hence it is statistically more probably that a fault will occur in a switching element. Secondly, if a fault occurs in an interconnect, it can be modelled by a corresponding fault in an adjacent switching element which will produce the same behaviour.

A faulty switching element is assumed to be non operational and can not perform any op-

erations including passing information through it. 147

It is assumed that mechanism exists to determine which switching elements are faulty. Each switching element in the network, must be aware of the status of the switching elements it is connected to.

One final restriction is placed on fault model. We assume that no faults can occur in either the first or last stage of the network. If a fault exists in the first stage, the two input terminals connected to the switching element will be isolated from the network. Similarly a fault occurring in the final stage of the network will isolated two output terminals from the network. If we wished to implement fault tolerance of faults in these stages, we could either multiplex input signals from multiple different switching elements, or increase the network size N' by 2, and have a redundant switching element in each stage.The isolated input or output terminals could then be ignored and routing from these terminals could be done by the extra switching element.

The restrictions placed on the fault models may seem to be overly restrictive, but they are in fact reasonable. Although the focus of this appendix is on single faults in the network, the result should be extendable to multiple faults.

Let us assume that switching element e_{rc} is faulty. We now define I_f,I_n,O_f, and O_n as subsets of I and O. I_f is the set of Input terminals who can route a messages to e_{rc}.I_n is defined as the set of input terminals who can not route messages to e_{rc}. Likewise O_f is defined as the set of output terminals which may have messages routed from e_{rc}. Likewise O_n is the set of output terminals which may not receive messages from e_{rc}.

$$I = I_f \cup I_n$$

$$I_f \cap I_n = \phi$$

$$O = O_f \cup O_n$$

$$O_f \cap O_n = \varphi$$

We also know that

$$|I_f| = 2^{c+1}$$

$$|O_f| = 2^{\lceil \log N' \rceil - c}$$

An example of I_f,I_n,O_f, and O_n can be found in the appendix Example 2

B.3.2Simple Bypass Mode

In the first routing algorithm developed, the goals was to find a routing algorithm which could find an alternative path from source to destination, if the T'_1 creates a path through the fault switching element, and T'_2 does not exists. In this section I discuss the simplest algorithm. If we know that no direct path exists from input to output because of the presence of a fault, we must find a path which takes a path greater than $\lceil \log N'''' \rceil$. These algorithms will all require that the network are capable of bidirectional routing. These types of algorithms are non minimal.

In our fault tolerant algorithm, we define three modes of operation

10. Normal Mode: In this mode a packets is routed towards the final destination as if no fault exists in the next work. Packets remaining in this mode until they encounter a faulty exchange element.

11. Bypass Mode: Once a packets attempts to be delivered to a faulty exchange element, it changes to BYpass mode. In this mode, the packet will bypass the faulty element by traversing a different path through the network. In this mode, the packet may or may not be heading towards any specific destination.

12. Resume Mode: Once a packets in Bypass mode is delivered to an exchange element who has a path to the original packets destination, a new control tag is determined (using either Theorem 1 or Theorem 2).

The simplest bypass mode (although not the most efficient) can be used for networks where N' is not a multiple of 4. In these networks.

13. If a packets is received from an output port of a switch element, it is sent out the other output port.

14. If a packet is received from an input port, it is sent out the other input port.

An example of the path taken to deliver a message in the routing algorithm is shown in Figure 75.

The delivery time in the presence of a single fault and no other network traffic is bounded by 2N'+2logN'

This network was simulated, and although a message was always capable of being delivered, the number of permutations which could be simultaneous routed is quite small when a messages needs to use bypass mode to route. This can be seen from the example in Figure 75. Only switching elements (0,2) and (0,1) are not blocked in the third stage of the network by the current message. It is possible for a single message to block all other permutations. Also the bidirectional nature of this network, also make it much more susceptible to deadlock problems.

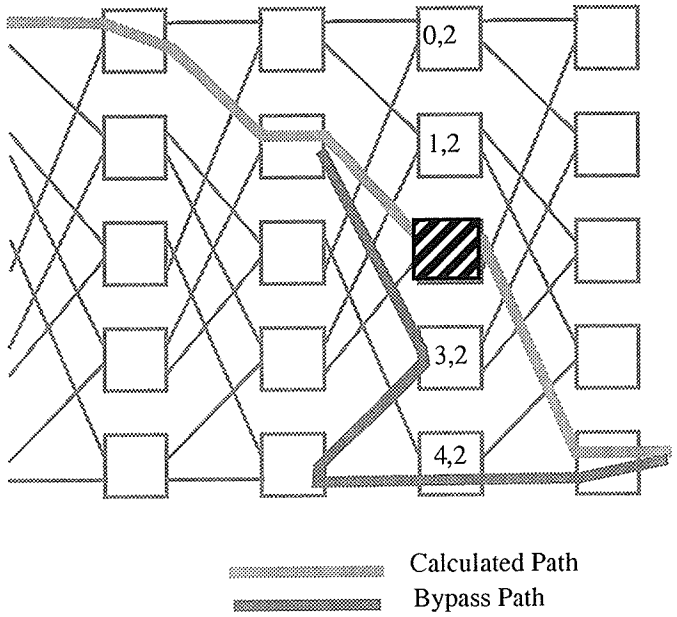


Figure 77 Routing around a fault in Bypass mode

B.3.3Two Pass Routing

In the previous section we discussed routing for networks in which the input and output terminals are independent (such as Processors connected to input terminals, memory connected to output terminals). This this section we investigate routing in networks where the input and output terminals are the same device, such as each terminal corresponds to a processor in a multiprocessor network. An example configuration is shown in Figure 78.

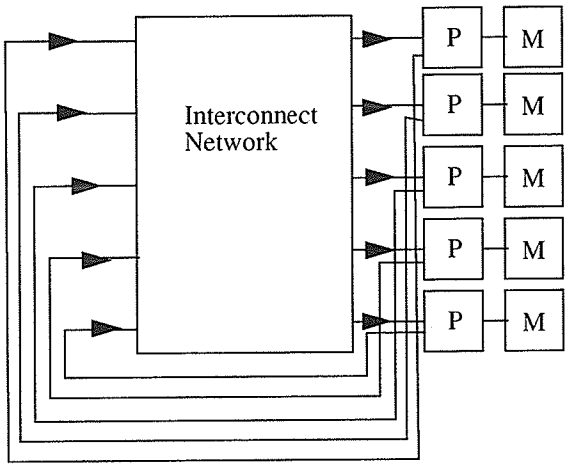


Figure 78 Multiprocessor Network

Routing in this type of network can be accomplished by attempting to route messages the same as in the fault free case. In the case where a message is unable to be delivered to its destination, only one path exists from source to destination, and it requires the fault switch, we route to an alternative destination. The destination which receives the message must then route the message to its final destination. To ensure that this algorithm will work, we must ensure that the destination which the message is routed during the fast pass, is capable of routing to the final destination. Specifically this means that it must not need to route through the faulty switching element.

Let us assume that we want to route a message m, from input i to output j. In the presence of a single fault in the network we can successfully route from i to j in one pass iff one of the following conditions is true:

$$I_i \in I_n$$

$$I_i \in I_f \text{and} O_j \in O_n$$

$$T'_2 \text{exists}$$

The routing algorithm based on this techniques attempts to route a message from i to j, using equations for T'_1 and T'_2..If a message can not be routed using these equations, which implies that

$$I_i \in I_f \text{ and } O_j \in O_f \text{ and } T'_2 \text{ does not exist.}$$

We need to select a intermediate destination k, such that

$O_k \notin O_f$ and $I_k \notin I_f$

these two conditions guarantee that the intermediate destination is reachable from the initial input, and can route to the required output. There are other intermediate destinations which will work, but they require the use of the second routing path from k to j.

Selection of the intermediate destination is straightforward. We know that:

$$I_f = \{i_i \mid (0 \le i < N') \text{ and } (i = \left\lfloor \frac{r}{2^r} \right\rfloor + j\frac{N'}{2^{r+1}}) \; j = 0..2^{r+1} \}$$

$$O_f = \{o_i \mid (0 \le i < N') \text{ and } (i = r2^{\lceil \log N' \rceil - r} + j) \; j = 0...2^{\lceil \log N'''' \rceil - r - 1} \}$$

This algorithm has worst case routing time of $2X\lceil \log N' \rceil + c$ where c is the time required for the intermediate destination to redirect the message between passes.

Simulations of this network and routing algorithm showed that it could successfully route messages in the presence of a single fault. Only a small number of messages actually needed two passes to route through the network. This algorithm is extendable to multiple faults.

B.3.4BB-ESSE Routing

In the previous section we discussed a networks where we could route in 2logN' time assuming that the input and output terminals are not independent. Often this is not the case, such as when we wish to use as MIN to connect multiple processors with multiple memory units[3], such as shown in Figure 78.

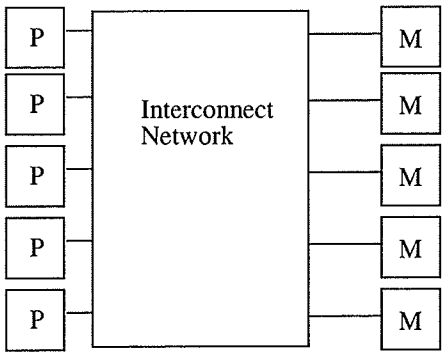


Figure 79 Multiprocessor Network

Although the routing techniques mentioned in the first two sections will route a message from source to destination in the presence of a fault, they require complex routing algorithms, not suitable for this type of environment. If we wish to allow a higher number of simultaneous connects

through the network, we must supply additional hardware. In this section we present one such extended network based on the Benes network[2]. This is not the only type of network which will utilize this concept, but it is discussed here as an example of one possible way to create a network with extended number of stages.

The BB-ESSE (Back to Back Even Sized Shuffle Exchange) network is constructed by placing two ESSE networks together, such the output terminals of both are connected. Such a network is shown in Figure 75.

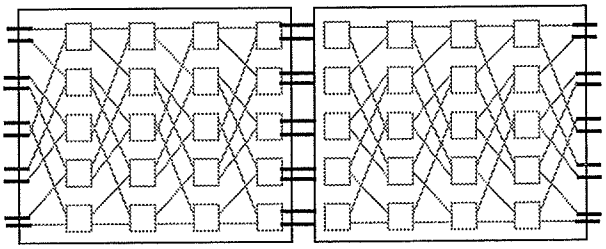


Figure 80 A 10 by 10 BB-ESSE Network

It is quite easy to route in this network in the presence of a single fault. Routing is accomplished by simply selecting an intermediate node in the network which is reachable from the input terminal (if fault is in forward ESSE network), otherwise select intermediate node which can route to output terminal from intermediate node (if fault is in the reverse ESSE network)., create a control tag which routes from source to the intermediate node and from intermediate node to output terminal using the routing equations developed in section 2. This network can easily handle multiple faults.

Simulations of this network showed that messages could always be delivered in the presence of faults, and a large number of permutations could be handled.

B.4 Summary

In this appendix we have extended the work on routing in Even Sized Shuffle Exchange networks. We first extended the network so that we could route in both forward and reverse directions, and showed how to calculate control tags to deliver messages

We also looked at various methods of incorporating fault tolerant routing techniques in ESSE type networks. First we showed a simple method by which a path could be found from source to destination terminals which bypasses faulty switching elements, Unfortunately this method was found give unacceptable performance in terms of number of simultaneous permutations that can be simultaneous handled. Alternative network structures were also discussed and simulation results showed better performance.

This work should be extended to show multiple fault coverage, and permutation accessibil-

ity.

B.5 References

[93]K. Padmanabhan, "Design and Analysis of Even-Sized Binary Shuffle-Exchange Networks for Multiprocessors", IEEE Transactions on Parallel and Distributed Systems, Vol. 2, No. 4, October 1991.

[94]F. T. Leighton "Introduction To Parallel Algorithms and Architectures: Arrays Tress Hypercubes", Morgan Kaufman, 1992.

[95]T. Lang, "Interconnections between processors and memory modules using shuffle-exchange network", IEEE Transaction on Computers, Vol. C-25,, pp. 496-503, May 1976.