PARALLEL COMPUTING WITH OPTICAL INTERCONNECTS

by
Mehrnoosh Mary Eshaghian

Technical Report No. CENG 89-02

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(Computer Engineering)

December 1988

UNIVERSITY OF SOUTHERN CALIFORNIA
THE GRADUATE SCHOOL
UNIVERSITY PARK
LOS ANGELES, CALIFORNIA 90089

This dissertation, written by

Mehrnoosh Mary Eshaghian

under the direction of her Dissertation Committee, and approved by all its members, has been presented to and accepted by The Graduate School, in partial fulfillment of requirements for the degree of

DOCTOR OF PHILOSOPHY

Barbara Solomon

Dean of Graduate Studies

Date December 15, 1988

DISSERTATION COMMITTEE

V. U. Prasanna Kumar

Chairperson

Gary F. Mitchell

B. Keith Jenkins

Irving S. Reed

To my parents

Table of Contents

List of Figures

List of Tables

Abstract

Optics has become an appealing alternative to wired interconnection on several levels of communication hierarchy within computing systems. Optical chip interconnections, unlike electrical, are insensitive to mutual interference effects, are free from capacitive loading and planar constraints, and can be reprogrammable. A major goal of this thesis is to understand the computational limits in using optical communication technology in VLSI parallel processing systems. Established methodologies for studying computational complexity are applied to obtain measures that reflect true implementation costs.

The computational lower bounds derived using the VLSI model of computation indicate that solution to communication-intensive problems requires either a large amount of chip area or time, both of which are costly. The first part of the thesis introduces an Optical Model of Computation (OMC) that uses free space optics as a means of interprocessor communication; thus reducing chip costs. The model allows unit cost communications, and can simulate one step of PRAM with no loss in time when the number of memory locations is equal to the number of processors. Reducing the number of processors by a factor of $O(\log N)$, simple algorithms are presented that run in $O(\log N \log\log N)$ time with a high probability, and in $O(\log^2 N)$ time deterministically.

Since OMC uses the space above and around chips for interconnects, OMC can be compared with three dimensional VLSI models in computational complexity. Any computation performed by a three dimensional VLSI organization having N processors with degree d, in time T, and volume V can be performed on OMC in volume v, and time t, where $dT/N \leq t \leq T$, and $Nd \leq v$. The thesis presents various parallel architectures as possible efficient upper bounds for v. Each one is designed to reflect the capabilities and limitations of the device technologies used for the redirection of optical beams. For example, an acousto-optic device, for which the redirecting capability is usually limited to only one dimension, is used to interconnect a linear array of processors. Holograms are used to realize a unit delay electro-optical crossbar. The crossbar's switching speed is in the order of nano-seconds.

Having developed the computational models, the thesis next focuses on applications in image processing and the implementation of AI problem solving techniques. A set of $O(\log N)$ pointer based algorithms for finding geometric properties of digitized images on an electro-optical mesh is introduced. The algorithms include optimal solutions for identifying and labeling figures, computing convexity properties, determining distances, etc. Another application is in the implementation of neural networks using a general purpose electro-optical crossbar which has the potential to interconnect each of the neurons to all the others. This architecture can be modified to operate asynchronously and to realize the data-flow model.

Part I

Background

Chapter 1

Introduction

Speedups due to technological advances in solid state electronic design are reaching theoretical limits. To get around these limits, researchers have considered concurrent processing of data as a promising alternative for achieving speedups proportional to the level of concurrency. During the past decade, many multiprocessor architectures have been proposed to obtain such speedups. However, the desired speedups have not been realized because of a limited understanding of issues in designing efficient parallel algorithms and in designing interconnection networks and their interactions. Recently, many parallel algorithms have been designed based on a theoretical shared memory model, the Parallel Random Access Machine (PRAM) [124], in which a unit-delay interconnection network is assumed.

In practice, interconnection networks introduce a delay factor in the implementation of parallel algorithms. The main issues in the design of such interconnection networks have been routing delay, communication bandwidth, hardware

cost, and ease of control. Traditionally electronic interconnects have been used. However, with advances in optical technology, it is likely that photonics will play an important role in parallel computation. This thesis focuses on possible realizations of unit-time interconnection network using free space optics.

1.1 VLSI Interconnection topologies

Research in the design of interconnection networks can be divided into two topological classes: static and dynamic[34]. In a static network, links between any two processors are passive and direct connections cannot be reconfigured between processors. In a dynamic network, links can be reconfigured by setting the switching elements in the network. Among many static topologies, those having small diameter are most attractive since the diameter of an architecture represents a lower bound on worst case communication delay between any two processors. A fully connected network has unit diameter. However, when implemented in electronic technology such as VLSI, its diameter becomes $\Omega(\log N)$ due to pin-out limitations of processors. Also, the VLSI layout area becomes too large to be practically implemented. Therefore, an appropriate alternative is to consider area efficient architectures which have $O(\log N)$ communication delay. Pyramid and Mesh of Trees are examples of such architectures [84, 71, 74]. Because of communication bandwidth constraints, simulation of some PRAM algorithms on

[2] A function $f(n)$ is said to be $\Omega(g(n))$ if there exist positive constants n_0 and c such that $f(n) \geq c \cdot g(n)$, for all $n > n_0$.

[3] A function $f(n)$ is said to be $O(g(n))$ if there exist positive constants c and n_0 such that for all $n \geq n_0$, $f(n) \leq c \cdot g(n)$.

these two architectures can lead to significant loss in time performance.

Dynamic networks are categorized in three topological classes: single-stage, multistage, and crossbar[34]. The switches in an $N \times N$ crossbar can be set in $O(\log N)$ time so that every input port can be connected to a free output port. However, an N input crossbar requires $\Omega(N^2)$ VLSI area, using the usual two dimensional VLSI model [113]. Several N input N output multistage networks are known which require $O(N \log N)$ switching elements; significantly fewer than a crossbar network [107]. Multistage networks can be divided into two major classes: rearrangeable and non-rearrangeable [10]. Non-rearrangeable networks can realize only a proper subset of all permutations. A Butterfly network is a widely used non-rearrangeable network. It has been shown that an arbitrary permutation can be realized by the butterfly network in $O(\log N)$ time, with a high probability [118].

Rearrangeable networks support any arbitrary permutation using appropriate switch settings. However, finding a switch setting to realize a permutation on a rearrangeable network can be time consuming; for example it can take as much as $O(\log^4 N)$ time using cube connected computer or a perfect shuffle computer with N processors [79]. Also, their layout area in the two-dimensional VLSI model, is not significantly superior compared to the area requirement of the N input crossbar. In fact, the well known omega network which is a non-rearrangeble multistage network and has $O(N \log N)$ switching elements, requires $\Omega(N^2/\log^2 N)$ VLSI area [71]. Therefore, realizing multistage interconnection networks in $O(N \log N)$ area does not seem possible unless one assumes that wires do not occupy any area, or if free space optical beams are used as a means

of interconnection.

In this thesis, we study parallel architectures that use free space optics as a means of interprocessor communications. Replacing electrical interconnects with optical beams has a significant impact on the performance of VLSI architectures [19, 47]. This fact arises from the following two important properties of free space optics. First, free space optical beams can cross each other without any interference. Secondly, the connections need not be fixed and can be redirected [12]. Therefore, using optical interconnects, one can design bounded degree VLSI architectures that can simulate a unit delay interconnection network.

1.2 Optical Interconnection Networks

A considerable amount of research has been done on replacing wires with optical waveguides on a VLSI chip [36, 48]. Since such an approach does not change the computational power of the classical VLSI model, it is outside the focus of our thesis. However, it is worthwhile to know that these techniques can be used in a finer level of integration, such as in the design of processing elements. Some of engineering merits of this approach are discussed in the following chapter.

Unique qualities of the optical medium are its abilities to be directed for propagation in free space and to have two optical channels cross in space with out interaction. These properties allow optical interconnects to utilize all three dimensions of space. The ultimate goal of reconfigurable interconnects is to be able to change the interconnect matrix as quickly and as freely as neeed. Such a capability will allow optical interconnection to improve upon many of

the functions presently implemented on a limited scale with electronics, such as routing data between processing elements based on data dependent decisions, and multiplexing and demultiplexing information.

One of the first attempts in using free space optics as a means of data-communications was [43]. In their hybrid GaAs/Si approach to data communication, a GaAs chip with optical sources was connected in a hybrid fashion (with conventional wire bond techniques) to a Si chip such that light was generated only along the edges of the Si chip. The sources were of the edge-emitting or surface emitting type. The optical signals were routed to the appropriate locations on the Si chip using conventional and or holographic optical elements. The Si chip contained detectors to receive the optical data streams generated by the sources. Since the detector-amplifier combinations were fabricated in Si, every computational component on the Si chip was capable of receiving data.

To explore this promising concept, it was extended to support efficient interconnection networks for massively parallel computing. One of the most recent contributions was [101]. In that, Sawchuck et al. described several possible bulk optical systems for implementing crossbar networks. Unlike the electrical crossbar, these crossbars provided unit time interconnectivity and had a slow switching rate. In this thesis, we propose a class of free space interconnection networks with unit time delay. The proposed crossbar has a switching time in an order of nanoseconds and is implementable with current technology.

1.3 Summary of Results

Part II concentrates on the proposed computational models. In chapter 3, an Optical Model of Computation (OMC) is presented along with its computational lower bounds for solving problems. This model uses the space above and around chips for interconnects, which makes it comparable to three dimensional VLSI models. Any computation performed by a three dimensional VLSI organization having N processors with degree d, in time T, and volume V can be performed on OMC in volume v, and time t, where $dT/N \leq t \leq T$, and $Nd \leq v$. Various parallel architectures are presented as possible efficient upper bounds for v. Each one is designed to reflect the capabilities and limitations of the device technologies used for the redirection of optical beams. A direct implementation of OMC is possible with an optical mesh using mirrors. A faster architecture will be found in an optical array using acoustic optic devices with broadcasting capability. A considerably less expensive and currently implementable design is the electro optical crossbar, which has a reconfiguration time in the order of nanoseconds.

The relationships to shared memory models and simulation algorithms are shown in chapter 4. A model of parallel computation motivated by the properties of free space optical interconnects is studied. In this model unit cost communication is possible, however there is a bottleneck in accessing memory modules. We will present a simple deterministic algorithm for efficient accessing of these memory modules. This algorithm simulates one step of an N processor EREW PRAM in $O(\log^2 N)$ time, and its probabilistic version has $O(\log N \log\log N)$ expected running time. The loosely synchronized point based techniques presented

can act as general subroutines in simulating PRAM algorithms.

The proposed model can be used to implement parallel solutions to a variety of problems in signal and image processing and non numerical problems arising in AI. Some examples are given in part III. In chapter 5, a set of optimal algorithms for finding geometric properties of digitized images is derived for fine grain electro-optical arrays. An efficient electro optical implementation of iterative solutions to sparse linear systems using holographic interconnects is also shown.

Another application of the proposed model is in providing efficient communication for non-Von Neumann types of computations. Implementation of non-Von Neuman computations such as the data flow mode of computation, and neural computing is the topic of chapter 6. Implementation of neural networks using a general purpose electro-optical crossbar which has the potential to interconnect each of the neurons to all the others will be shown. We will also show that similar architecture can be modified to operate asynchronously for realization of a data flow model. Our results substantiate the preference of optical interconnections over an electronic medium as a means of interprocessor communication.

Chapter 2

Optics and VLSI

Initial research on integrated optical devices and circuits was stimulated because of a perceived need for compact, rugged, and economical fiber-optic repeaters which are insensitive to external thermal and mechanical variations, and immune to electromagnetic interference from surrounding electric fields [8]. It was predicted that significantly higher usable bandwidths could be achieved by using optical frequencies for all data transmission functions. Batch fabrication technology would provide high reliability and small size. These advantages clearly define the great potential of integrated optics.

Since the early experimental attempts at integration of optical devices for data transmission, in late 1960's, the field has evolved slowly, with a few niche applications in higher speed data processing. Today, optically simple and functionally complex circuits are being manufactured for signal processing applications. These circuits usually offer the system designer a compact, economical, high-bandwidth

device for Fourier transform and correlation operations that would normally require complex electronic circuits ranging in size from several circuit boards to main frame computers.

In this chapter, we first review some of the fundamentals of integrated optics. We then continue with a discussion of merits and components of optical interconnections in VLSI.

2.1 Preliminaries

The physics of light is a very interesting subject. Even people with no background in physics have probably heard about the dual nature of light: on one hand it is composed of elementary particle-like quanta, called photons, and on the other it may be regarded as a wave [32]. For our purposes it will suffice to resolve the ambiguity by considering each photon to be a small wave packet. The light wave is then the sum of many photons. In the context of optical computers, we are usually interested only in the wave-like properties of light. The fact that this wave is the sum of many photons is usually irrelevant. We therefore turn to a discussion of the properties of waves in general, and electro-magnetic waves in particular [31].

Frequency is the most basic property of a wave. Frequency is the number of cycles a wave completes in one second. Electro magnetic waves come in a wide range of frequencies. Visible light occupies a rather narrow band between 10^{14} cps (red) and 10^{15} cps (violet). The reason both nature and optical computers

use this range is that many materials transmit and absorb radiation in these frequencies better than in other frequencies, through means of interactions between photons of radiation and the electrons in the material.

Another parameter of interest when talking about light is its wave length. This is the distance the light propagates during one cycle. Therefore the wavelength for a given frequency depends upon the speed of propagation. Speed in turn depends upon the medium through which the radiation is propagating. The index of refraction is a characteristic of the medium that governs the speed of light as it passes through. The refractive index of the vacuum is 1, and the speed of the light through it is approximately 300,000 Km per second. Other materials have a higher refractive index, and the speed of light in them is proportionately lower. When light passes from one medium to another, its change of speed causes a change of direction, called refraction. Lenses and other optical devices use this phenomenon to change the direction of rays of light. Such devices are actually just specially shaped transparent objects with refractive indexes that are different from their surroundings.

Two additional parameters are needed in order to describe a wave. The first is its amplitude, which is the height of the wave. The square of the amplitude is called the intensity of the wave, and conveys the number of photons in the wave. Since each photon is a quantum of energy, the light intensity also indicates how much energy is propagating with the wave. The second parameter is the wave's phase, which stands for the part of the cycle that the wave is in. Absolute phases are of no interest; only the phase of one wave relative to another is of consequence. When we describe a wave by harmonic functions, e.g. sinusoid, the

cycle is 2π. A phase difference of $\pi/2$ between two waves then means that one of them lags behind the other by one quarter of a cycle.

When two waves combine, they are said to interfere. If the waves have the same frequency, the result depends on their relative amplitudes and phases. If the crests in one wave coincide with the crests in the other, each wave cancels out the other This is called constructive interference. If the crests in one match the troughs in the other, the waves tend to cancel out. This is destructive interference (Figure 2.1).

The photons in Figure 2.1-b (constructive interference) are in phase with each other. Therefore their sum is a wave that can be described by a harmonic function such as $\sin(x)$. We call such waves coherent light. In many situations the phase of the photons, and thus of the light wave, changes in a random manner. When this happens, we say the light is incoherent.

2.2 Photonic Fundamentals

The properties of the photons explained in the previous section are key design issues in optical computing. The ultimate limit on optical computing speed is the velocity of light in the material, which may be on the order of 10^{10} per second. Photons may or may not interfere depending on the coherence properties of the photons in the individual beams. Two light beams can intersect in free space or in a dielectric waveguide with no interference occurring. At the same time, photons from coherent sources can interfere constructively or destructively when combined after traveling different paths. Both constructive and destructive

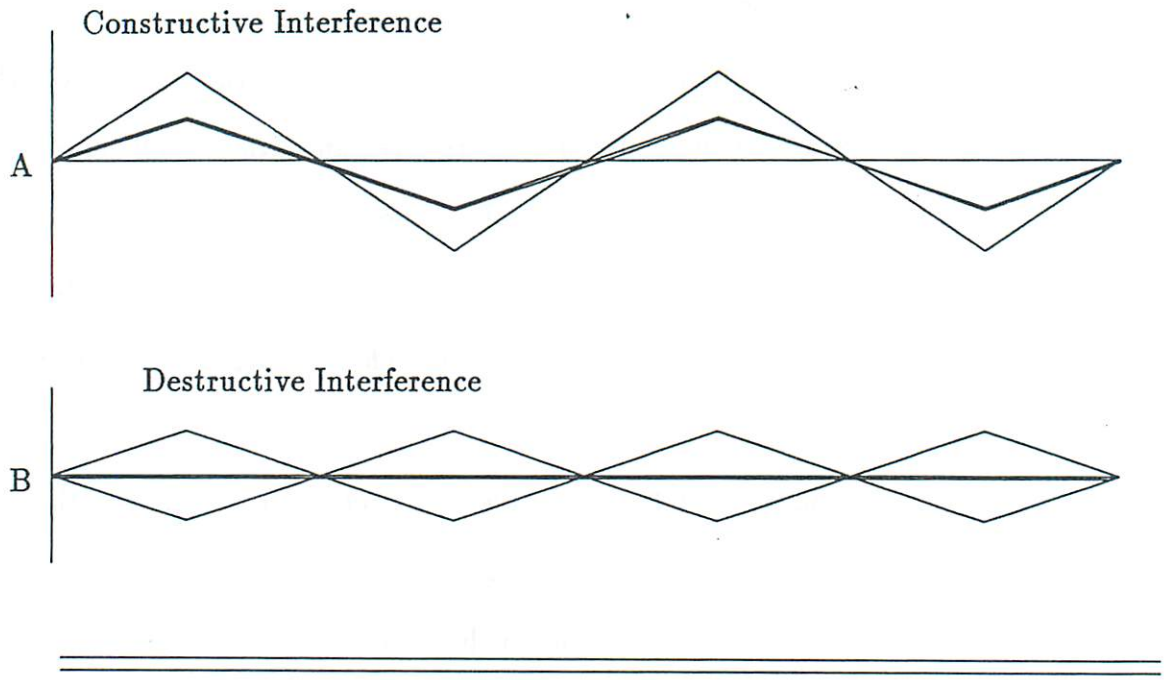


Figure 2.1: Interference between two waves

behaviors are advantageous in optical computing and signal processing devices. In contrast, electrons interact with each other through their electric fields and repel or attract other charged particles.

A third fundamental property of the photon used extensively in optical devices is that the energy of the photon is related to its frequency through Planck's constant. This energy for the photon is generated by the transition of an electron or molecule between energy states in a material which has been excited. Thus, the designer can select the frequency of the photon by choosing the proper material (e.g., semiconductor materials).

Transmission of information via photons requires no conducting materials and relies solely on low-loss dielectrics or free space. Over short distances (< 1 meter),

the bandwidth of either the optical waveguide or the free space path is limited only by optical modulator technology, whereas the bandwidth of an electronic system is limited by the inductance and capacitance of the metallic transmission path. Although this bandwidth advantage may be exciting, there is a price to pay in larger device dimensions. Waveguide dimensions will be on the order of an optical wavelength (1 micron), and most devices are a factor of 10 or more larger. Aside from the above factors, meaningful size and speed comparisons between electronic circuits and integrated optical circuits must be made on the basis of functional operations (i.e., correlation, convolution, etc.) rather than on individual circuit elements. The speed and parallelism of optical components may lead to smaller packaged circuits even though individual components are larger than submicron VLSI devices.

Finally, photons can be refracted by a lens or diffracted by a slit or pinhole. They may also be scattered by impurity particles and absorbed or refracted by certain materials. The fact that some of these properties can be explained by electromagnetic wave theory, while others cannot, led to the field of quantum mechanics and, in particular, to quantum optics. While more detailed discussion of the properties of photons is beyond the scope of this thesis, it is sufficient to point out that photons exhibit a number of properties different from electrons and provide many new and entirely different possibilities for optical computing. Thus, photon computing should not be viewed simply as another method for duplicating existing computer architectures and operations. It should be looked upon as a unique technology with new potential for computing.

2.3 Why Optics in VLSI

It has been demonstrated that optical techniques provide a much higher density for a given bandwidth than electronic techniques. Using free space or wavelength propagation, it is possible to take advantage of the high density bandwidth product available in the optical domain. In addition, using integration of opto-electronic devices, it is possible to communicate with the interior of the chip rather than confining the I/Os to the chip boundary. Following are some of the main limitations of electrical interconnects which are absent from optical integrated links [19, 67].

Several characteristics of present hardware techniques limit the density of electrical interconnects. One limitation is that the edge of the chip is reserved for I/O functions. Another is that electrical interconnects are confined to pseudo-planar structures (e.g., printed wiring boards, backplanes). The phenomenon of crosstalk is a Fundamental limitation on spacing between individual interconnects. This density issue is aggravated with increases in speed in individual devices. As speeds increase, sensitivity to crosstalk through the electrical interconnect also increases and the required distance between devices decreases to ensure that the signal propagation time is less than the clock period.

Obviously, as density increases, the spacing between lines decreases and it is necessary to reduce the cross-sectional area of the conductors in order to place more interconnections in a given volume. Since the dc series resistance of a conductor is directly proportional to its cross-sectional area, this scaling down will result in a larger voltage drop across the interconnect and an increase in

the power required to drive the line. This problem is compounded at higher frequencies because the skin effect reduces the effective conductor area even further. Therefore, as the bandwidth, density, and length of the lines increase, the amount of available power becomes a limiting factor in system performance.

2.4 Transmitting Channels

There are two approaches to defining the optical interconnect transmission medium: guided-wave interconnect and free space coupling. In guided wave interconnect the optical data are transmitted and distributed via a medium that localizes the optical energy and distributes it to the required destination points. Examples are optical fibers, light pipes, and planar waveguides. Free space coupling involves point to point or broadcast transmission of the optical energy through space. Both guided wave and free space coupling have trade offs that will affect the design of the system interconnect topology.

Several design methods have been investigated. All promise use of the space above and around chips for interconnects. With this added space and the high density potential of optical interconnects, it is possible to increase the number and bandwidth of interconnects within a computing architecture, resulting in faster systems.

2.4.1 Fiber Optics

An optical fiber is a conductor, or waveguide, for light. The propagation of light in (bent) optical fibers is explained by the phenomenon of total internal reflection

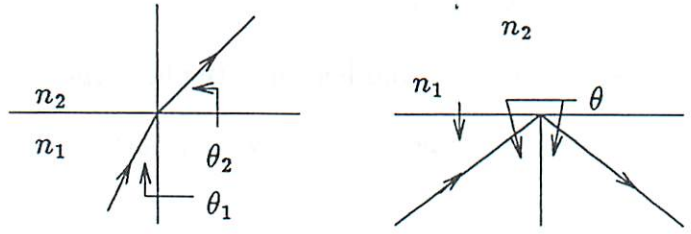


Figure 2.2: Refraction of rays

(see Figure 2.2). The fiber's cross-section has two parts: a core and a cladding. The difference between them is that the core has a larger index of refraction. We denote these indices by n_1 (core) and n_2 (cladding). A beam of light that hits the core/cladding interface at an angle θ_1 (from the core side) is refracted, and emerges at an angle θ_2 (on the cladding side). These angles satisfy the relation

$$n_1 \sin\theta_1 = n_2 \sin\theta_2$$

Since $n1 > n2$, the $\theta_1 > \theta_2$ (all angles are in the first quadrant). If θ_1 is larger than the critical value of arcsin $(n2/n1)$, the formula cannot be valid, because it would require that $\sin\theta_2 > 1$. Indeed, for these angles the beam is not refracted. Instead it is reflected back into the core. When a beam of light propagates along an optical fiber, it actually follows the zig-zag path that results from internal reflection each time it hits the core/cladding interface (see Figure 2.3)[66].

The fiber medium is a low loss, low dispersion channel. Over the short links discussed here (inches or less), the channel is virtually bandwidth unlimited. The transport of baseband information over a fiber can be accurately thought of as

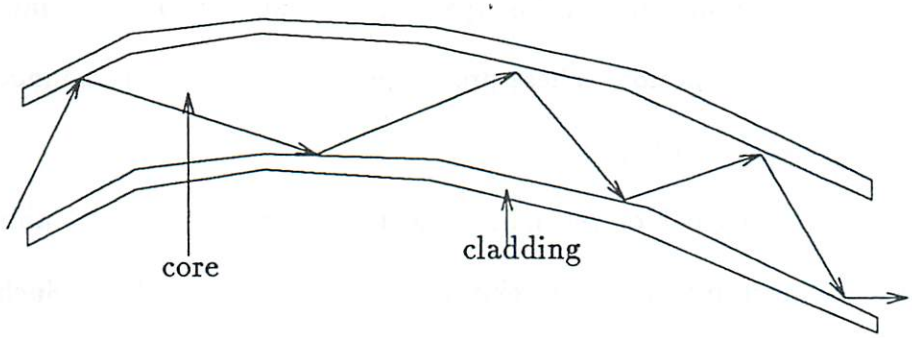


Figure 2.3: Light advance within an optical fiber

the modulation of a carrier oscillating at approximately 2×10^{14} Hz.

Fiber optic transmission techniques are now well developed and provide a medium for high bandwidth, long haul data transmission [48, 47]. Crosstalk and electrical interference problems, which are expected to increase with circuit densities and speeds of future digital circuits are absent with optical fibers. The loss and drive power are independent of the length of the fiber for distances of interest. The optical approach does not require high speed ground planes, can make circuit layout more flexible, and should be less sensitive to reflection.

Although fiber optics is one important subset of the set of photonic solutions, it is by no means the only (or necessarily the best) solution.

2.4.2 Free Space Interconnects

Guided waves have handicaps that severely limit their use, in particular, high attenuation, high bending loss, and constraint to a plane if fabricated in an

integrated optics form. A unique quality of the optical medium is its ability to be directed for propagation in free space and to have two optical channels cross in space without interaction. These properties allow optical interconnects to utilize all three dimensions of space.

The ultimate goal of reconfigurable interconnects is to be able to change the interconnect matrix as quickly and as freely as needed. Such a capability will allow optical interconnection to improve upon many of the functions presently implemented on a limited scale with electronics, such as routing data between processing elements based on data dependent decisions, and multiplexing and demultiplexing information. Potential applications for reconfigurable holographic optical elements are so compelling that investigation of alternatives for programmable interconnections will be of great interest in the future.

This thesis presents various programmable interconnection networks. Programmable interconnection networks are not yet developed enough to form an integrated environment. Alternatively, bulk realization of reconfigurable free space optical interconnects have been studied [101]. What is common to both integrated and bulk implementations is the use of gratings. In current integrated circuits the gratings are fixed and cannot be reconfigured. In bulk systems design, they can be reconfigured using various available spatial light modulators.

2.5 Grating

A (thin) grating is a set of fine, straight parallel lines on a transparency [63]. Like mirrors, gratings may be used to change the direction of a beam of light.

However with a grating, the beam is also split into a number of beams that go in different directions. In each of the beams, the exact change in direction depends on the wavelength of the light. Thus a grating may be used to obtain spectral analysis of polychromatic light.

When a coherent beam of light impinges upon a grating, it can be considered as equivalent to a set of coherent, long, linear light sources. The light from these sources interferes. In certain directions, on the path differences for light coming from adjacent sources (i.e. adjacent lines) is an integer multiple of the wavelength, and constructive interference occurs. These are the directions in which we get beams of light. In other directions destructive interference occurs, and the waves tend to cancel out. Generally, the directions in which constructive interference takes place must satisfy (see figure 2.4).

$$d \sin \theta = n\lambda \text{ (Bragg's Law)}$$

where d is the grating spacing, λ is the wavelength, n is an integer, called the diffraction order.

Note that this assumes the incident light beam is perpendicular to the grating so that all the lines oscillate in phase.

If the grating is thick, i.e. it is a set of parallel plane surfaces, the situation is slightly different. A light beam impinging on such a grating is reflected by each surface individually. If the angle between the beam and the surfaces is just right, constructive interference will occur between the reflections. The result will be a very strong diffraction. At other angles the diffraction will be very weak. The angle at which strong diffraction occurs is called the Bragg angle, and it satisfies

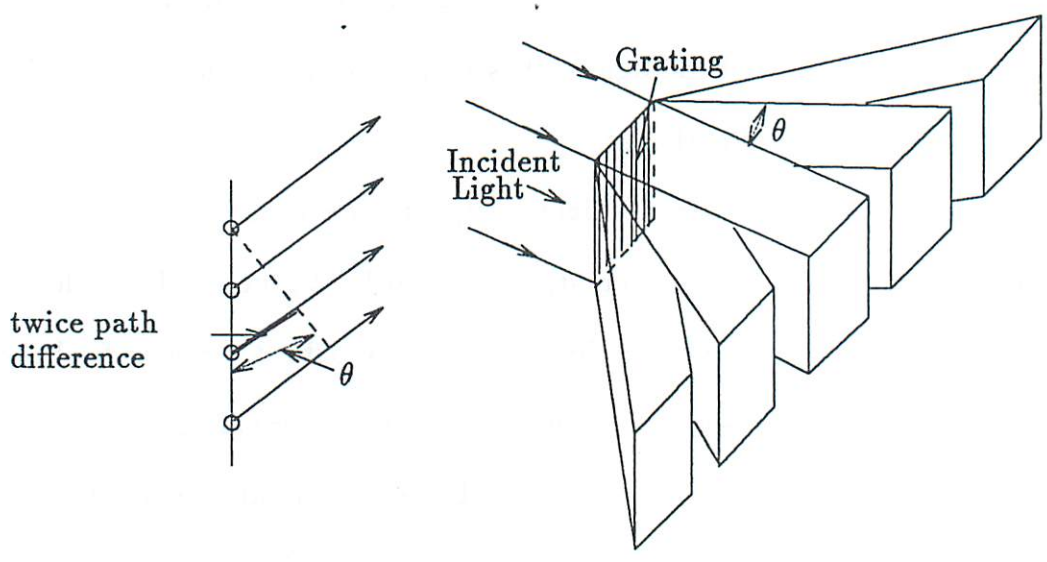


Figure 2.4: Bragg Diffraction

$\sin\theta = (n\lambda/2d)$. This phenomenon is of major importance in volume holography and some acousto-optical devices, as well as in the diffraction of x-rays from crystals.

2.5.1 Acousto-Optical Elements

Gratings need not be stable and tangible like most other devices; in fact it is quite common to use an acoustic wave as a grating. An acoustic wave propagating in a crystal causes a periodic deformation of the crystal. This deformation usually results in a periodic change of the refractive index of the crystal. The velocity of the acoustic wave is always much lower than that of light waves. Therefore we may say that to a good approximation a light wave passing through the crystal

feels a standing periodic change of the medium- that is, a grating. This grating causes diffraction of the light, just like any other grating would. Such phenomena in which light is scattered by interaction with acoustic waves are called Brillouin scattering.

2.5.2 Holographic Optical Elements

Holograms can be written on silicon in the following way. The photographic film hologram is contact printed onto a layer of photoresist on a silicon substrate. The photoresist is developed and the silicon is etched by chemical means. After removal of the remaining photoresist, a reflective surface-relief hologram is present in the surface of the silicon. Diffraction efficiencies of 18 to 20 silicon surface-relief holograms is possible.

Assume a single source of light. A simple hologram, similar to a diffraction grating, can deflect its light in any desired direction (depending on the orientation of grating), and focus it on a sink. A more complicated hologram can create multiple images of the source, each on a different sink. In the general case we have a set of sources, a set of sinks, and an arbitrary mapping of sources to sinks. In order to implement this mapping, the hologram can be divided into sub-holograms, one for each source [103, 59, 58]. Each sub-hologram is illuminated by one source only, and it images that source on the appropriate sinks. (see figure 2.5).

Free space interconnections utilize three-dimensional space [12]. There are various ways in which to set up the sources, sinks and hologram [43]. The most compact scheme is to have the sources and sinks arranged in a plane, with a

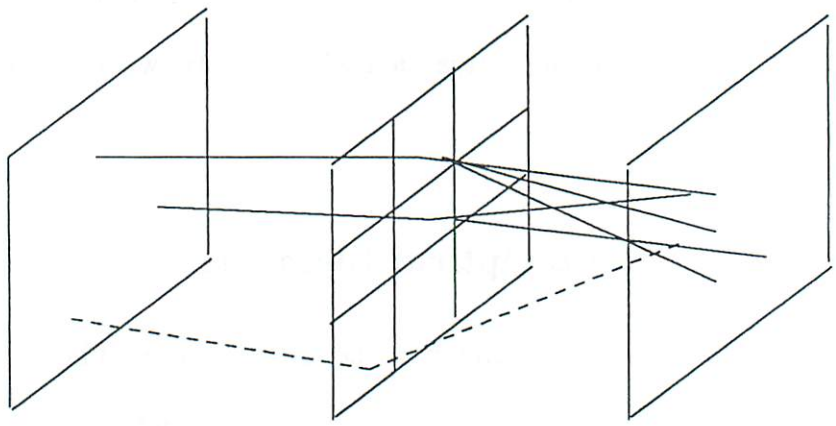


Figure 2.5: A holographic interconnection element

reflection hologram suspended above them (see figure 2.6). Other schemes use transmission holograms instead of reflection ones, and then redirect the light with mirrors. If the hologram is implemented by an Spatial Light Modulator (SLM), rather than being fixed, the interconnection pattern may be changed dynamically [69, 43].

Optical noise in the interconnections will arise from two aspects of the Holograms. First, to whatever extent the hologram is not perfectly efficient, the undiffracted light produces a diffuse unfocused background light level at the detectors. Second, all holograms produce some amount of stray light, resulting from stochastic effects in the recording process and, in the case of computer generated holograms, from quantization effects in the encoding. To maximize the signal reaching the detectors and to minimize optical effects on surrounding circuitry, the spot size should closely match the size of the detectors, i.e. about 10 by 10

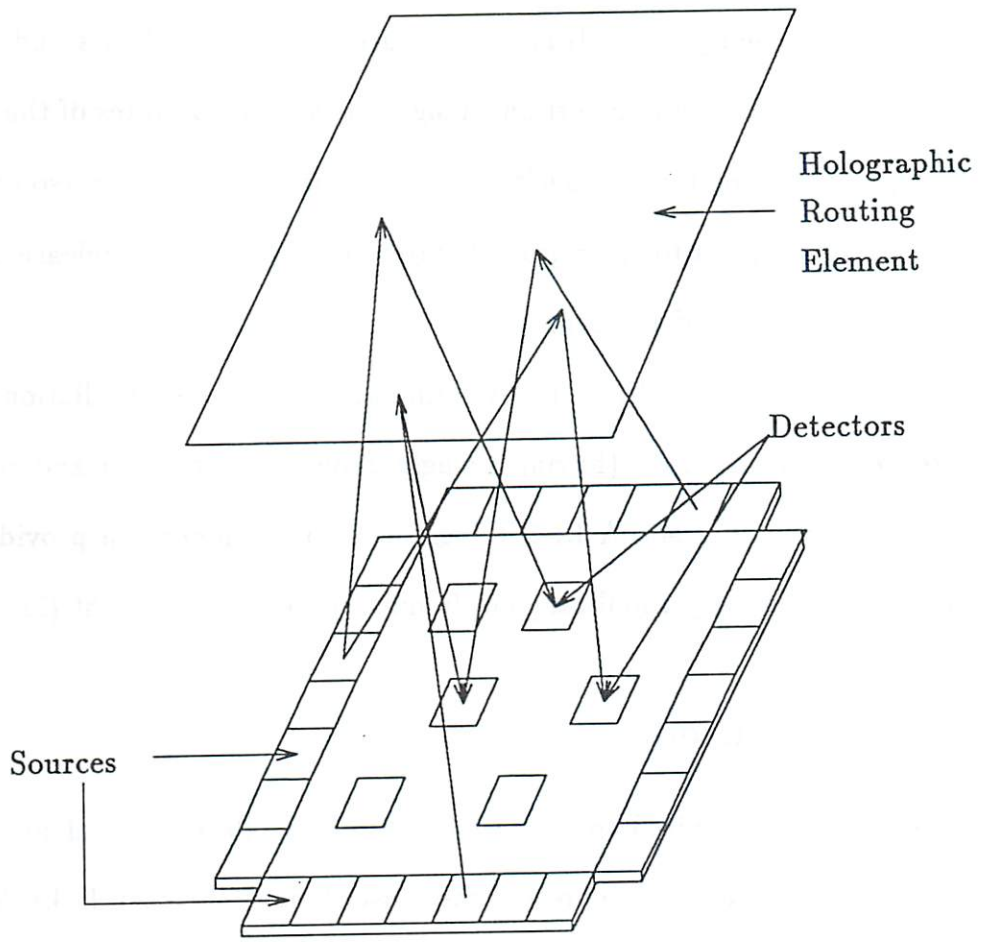


Figure 2.6: A possible holographic configuration

μm

2.6 Sources

The properties of an optical system depend to a large degree upon the source of the light that is being used. Today, two sources dominate: LEDs and LASERs. Their operations, depend on certain changes in the energy states of the electrons in the light-source material. Specifically, electrons in high-energy (so called "excited") states go down to their ground states. In doing so they release an energy quantum in the form of a photon.

A LASER (Light Amplification by Stimulated Emission of Radiation) outputs an intense, monochromatic (having a single frequency), coherent and very directional beam of light [104]. A LED (Light Emitting Diode) can provide narrow band (having relatively small range of frequency), incoherent light [11, 77].

2.6.1 Laser Diodes

Light sources are required to convert electrical signals to optical form. Laser based optical interconnects represent the most obvious choice of links. The compact size and low electric power requirements of laser diodes will ease the problems associated with the assembly and integration. One significant advantage of the laser diode is its potential for high modulation frequency, which has been demonstrated as high as 18 GHz.

With direct drive, the laser diode can be full on/off modulated at multi-Gb/s rates. Small signal modulation has been reported beyond 10 GHz. The total

average current required to drive the laser diode ranges from 15 mA to 45 mA. Total average power dissipation for a laser diode, then will range from 40 mW to 180 mW. Typical dimensions of a laser diode is 200 by 200 by 75 μm [48, 47].

2.6.2 Light Emitting Diodes

LEDs are highly reliable devices with no threshold requirements, are relatively simple to fabricate, and can be easily fabricated as edge or surface emitters. Performance limitations include low quantum efficiency, a high drive power requirement and low speed [48, 47]. Semiconductor laser diodes although similar in structure to LEDs are highly efficient. When they are operated above threshold the internal quantum efficiency approaches 100% for the most efficient class of laser sources developed.

2.7 Detectors

Unlike a light wave telecommunications receiver, an optical interconnect receiver is basically a high speed opto-electronic transducer. For optical interconnect applications, the main object is to receive the incoming optical data stream, transform it to electronics, amplify (and filter) it, and redigitize it.

The diameter of a detector is approximately 40μm. GaAs detectors with rise times on the order of 30 to 40 ps have been made [67]. There are several alternatives for GaAs detectors: the p-i-n diode, the avalanche photodetector (APF), and the Schottky barrier photodiode.

2.8 Semiconductor Integration

Monolithically integrated opto-electronic circuits that incorporate electronic and optical devices on the sam substrate have potentially important advantages over conventional hybrid circuits. The monolithic integration eliminates the excess capacitance and inductance associated with bonding pads and wires. The resulting circuit operates faster than its hybrid counterpart, with lower noise and less power. Integration also eliminates the need for broad band impedance matches. It is expected that integrated lasers and detectors can be realized in high density form with dimensions of 50 to 200 μm for laser cavities and photodetectors. Current monolithic implementations of optical interconnect technology involve either sources and detectors integrated with electronic circuits on GaAs or GaAs grown on Si. Both of these approaches are difficult to implement at present [36, 18].

2.8.1 Silicon Substrate

Silicon based integrated optics are nearly monolithic: only the source is in GaAs, and all the rest is in Si. Silicon offers a stable base for electronic circuitry; silicon VLSI technology is well established and has a large established industry [36]. State of art MOS circuits are quite fast, with cycle times up to 3 GHz reported in the literature. Long wavelength optics require epitaxial techniques on either silicon or gallium arsenide. Where is desirable to interface directly with long distance optical fibers (such as in telecommunication environment which operates at long wavelengths), this means that the processing required to build monolithically integrated optical components is as difficult in GaAs as in silicon.

Finally, silicon provides a lower-cost substrate for components that do not need extreme speed. These components form the vast majority of most systems.

2.8.2 Gallium Arsenide

Monolithic Gallium-Arsenide (or rather various alloys of GaAl-As) is probably on its way to becoming the preferred choice. The reason is that this is the only system of materials that enables all the necessary active devices-lasers, modulators, switches and detectors- to be fabricated together. In particular, the most efficient and useful light source today for fiber optics and integrated optics is the GaAs double heterostructure injection laser [110, 99]. However differences in the detailed requirements for the laser versus the waveguide applications remain to be resolved before a true monolithic technology emerges.

GaAs ICs and GaAs opto-electronic devices, such as diode lasers and photodiode detectors are needed to construct multigigahertz optical transmitters, receivers, transceiver, repeater, and switches.

2.8.3 Hybrid Integration

Eventually, the sources must be brought onto the electronic chip either through fabrication of the electronics on GaAs or by fabrication of GaAs sources on Si substrates [18]. The best waveguide active devices, eg. modulators and switches, are those fabricated on Lithium Niobate substrates. However the sources used are usually based on GaAs, while the detectors employ Si. Hence the hybrid nature of this system, as an example.

2.9 Limitations

Whereas in VLSI the minimal feature size is determined by the technology and by the microscopic characteristic of the materials, in optics the minimal feature size corresponds to the wavelength of the light [68]. The size of optical elements must be at least a few wavelengths: light simply cannot be confined to a cross section of much less than a wavelength, i.e. approximately 1 μm. At the moment it so happens that VLSI technology produces features that are about of this size too, however it is already obvious that it will be impossible to miniaturize optical devices to the same degree that is possible in electronics, as it is expected that electronic devices will be reduced by at least another order of magnitude before reaching their limit [106].

Part II

Computational Models

Chapter 3

Optical Models of Computation

The relation between the speed and size of VLSI circuits was explored using the methodology of complexity theory in [113]. As the first step of this methodology, an accurate and precisely-defined model of a VLSI chip was devised. This model captures the two-dimensionality of VLSI, in that transistors are laid out on the surface of a piece of silicon, and there are only a few layers of metal available for interconnections. The fundamental result that clearly defines the limits of VLSI is due to the fact that the amount of time (T) required to solve a problem on a VLSI chip is at least equal to the number of cycles required to pass the minimum required information (I) over the available bandwidth across the mid line of a VLSI design having area (A). This leads to $AT^2 = \Omega(I^2)$. In this chapter, we introduce an abstract optical model of computation to explore speed size relationship in using free space optical beams, as opposed to wires, for means of intercommunications. This model accurately represents currently implementable

optical network of processors. Hence, the derived lower bounds on its computational efficiency gives us a tool to analyze the optimality of various physical implementations of OMC, in solving problems. In the first section lower bounds for computation power are derived, and in the following section three architectures representing upper bounds on the volume requirements of the model are shown.

3.1 Lower bounds

In this section, we define an Optical model of computation which is an abstraction of currently implementable optical and electro-optical computers. Similar to the VLSI model of computation [115], this model can be used to understand the limits on computational efficiency in using optical technology. We show that minimum volume requirement of an optical model of computation is same as the minimum VLSI area in the VLSI model. Using information transfer argument, we also show a methodology to determine the minimum volume requirement of an electro-optical system for solving a problem.

3.1.1 An Optical Model

An optical model is shown in Figure 3.1. More formally this model is defined as follows:

Definition 1 *An optical model of computation represents a network of N processors each associated with a memory module, and a deflecting unit capable of establishing direct optical connection to another processor. The interprocessor communication is performed satisfying the following rules similar to [4]:*

1. *At any time a processor can send at most one message. Its destination is another processor.*

2. *The message will succeed in reaching the processor if it is the only message with that processor as its destination, at that time step.*
3. *All messages succeed or fail (and thus are discarded) in unit time.*

To insure that every processor knows when its message succeeds we assume that the OMC is run in two phases. In the first phase, read/write messages are sent, and in the second, values are returned to successful readers and acknowledgements are returned to successful writers. We assume that the operation mode is synchronous, and all processors are connected to a central control unit. The above definition is supplemented with the following set of assumptions for accurate analysis.

1. Processors are embedded in the Euclidean plane. This is referred to as the processing layer.

2. Each of the processing/memory elements occupies unit area.

3. Deflectors are embedded in the Euclidean plane. This is referred to as the deflecting layer.

4. Each deflecting unit occupies at least one unit area.

5. The deflecting layer is collinear to the processing layer.

6. I/O is performed at I/O pads. Each I/O pad occupies unit area.

7. The total volume is the sum of the volume occupied by the processing layer, the deflecting layer, and the space for optical beams.

8. The intercommunication is done through free space optical beams.

9. Time is measured in terms of number of units of clock cycles.

10. An optical beam carries a constant amount of information in one unit of time, independent of the distance to be covered.

11. A deflector is capable of redirecting an incident beam in one unit of time.

12. A processor can perform a simple arithmetic/logic operation in one unit of time.

13. The time, T for computation is the time between the arrival of the first input to the departure of the last output.

To be able to compare our results with those on VLSI model of computation [113], without loss of generality, assume that there are N processors placed on a $N^{1/2} \times N^{1/2}$ grid called the processing layer. Similarly, there are N deflecting modules on a layer above the processing layer, called the deflection layer. The interconnection beams are established in the free space between these two layers, as shown in Figure 3.1. Hence, the amount of data that can be exchanged in a cycle between two sets of processors (two way information transfer rate) is N. The time (T) required to solve a problem is the number of cycles required to exchange the minimum required information (I). This leads to :

$$AT = \Omega(I)$$

where A is the area occupied by the processing layer.

A related model is VLSIO [7], which is a three dimensional generalization of the wire model of the two dimensional VLSI with optical beams replacing the wires as communication channels. Compared to the three dimensional VLSI model of computation [95], our model is more resource efficient. The simulation of many parallel organizations using the OMC requires considerably less amount

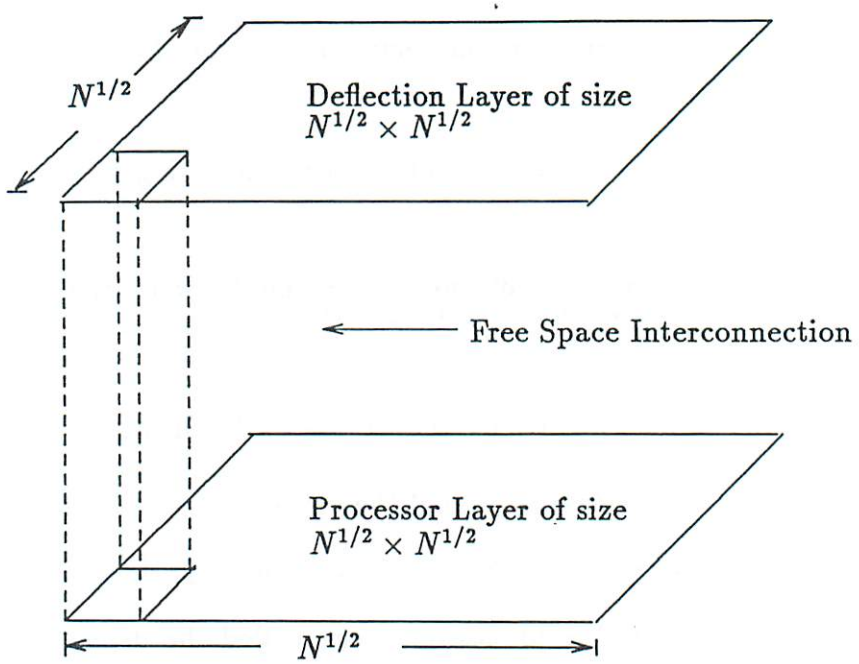


Figure 3.1: Optical Model of Computation

of volume than their layout in the three dimensional VLSI model. For example, the layout volume of a N processor hypercube can be reduced from $O(N^{3/2})$ to $O(N \log N)$ when using OMC with mirrors as deflectors. The following result can be stated:

Proposition 1 *Any computation performed by a three dimensional VLSI organization having N processors with degree d, in time T, and volume V can be performed on OMC in volume v, and time t, where $dT/N \leq t \leq T$, and $Nd \leq v$.*

The upper bound on t is obvious. Its lower bound also can be simply obtained by multiplying T by d/N which is the maximum speed up factor that can be obtained due to its unit time interconnection medium. The lower bound on v is

obtained by the minimum area requirement for having d deflectors for each of the processing elements. In the next sections three different parallel architectures are presented as possible efficient upper bounds for v.

3.1.2 Optical Volume, VLSI Area and 1-way Information Transfer

The minimum VLSI area requirement for computing a problem is related to the *lower bound* on the *1-way information transfer* [126, 82]. In this section, we briefly discuss the 1-way information transfer for computing *single output function* and then extend the argument to *multiple output functions*. We also relate this 1-way information transfer to the Optical volume requirement of an electro-optical implementation of OMC to solve a problem.

The following abstract setting has been shown to be useful in estimating the minimum VLSI chip area [82, 112]. Two sets of processors $P1$ and $P2$ each receive $\frac{n}{2}$ bits of an n input function f to be computed. The input partition can be denoted by (π_1^i, π_2^i) where $\pi_1^i(\pi_2^i)$ are the inputs known to $P1(P2)$ and $\pi_1^i \cap \pi_2^i = \phi$. Also $|\pi_1^i| = |\pi_2^i| = \frac{n}{2}$. The rectangle corresponding to this partition is defined as $M \times N$ where $M(N)$ = set of all values known to $P1(P2)$. It is clear that $|M| = |N| = 2^{\frac{n}{2}}$. In the 1-way information transfer model, $P2$ computes f and outputs the result. Hence, given an input partition, some information based on the input bits of π_1^i are transferred to $P2$ in order to complete the computation. The minimum *information transfer* from $P1$ to $P2$ to compute f over all possible input partitions is denoted by $I_1(f)$ and is defined as follows:

$$I_1(f) = \underset{input\ partition}{Min} \left\{ \begin{array}{c} worst\ case\ information\ transfer \\ from\ P1\ to\ P2 \end{array} \right\}$$

Two rows $i_1, i_2 \in M$ in the *computational rectangle* are said to be *distinct* if $\exists$ a $j \in N$ such that $f(i_1, j) \neq f(i_2, j)$. If $d(f)$ is the minimum number of such *distinct* rows over all possible input partitions, then $I_1(f)$ is equal to log $d(f)$ [125]. The area requirement A of any chip computing f is $\Omega(I_1(f))$ [126, 82].

The above 1-way protocol for single output function can be easily extended to multiple output functions by introducing a suitable output partition over the output functions. Let $F = \{f_1, f_2, \ldots, f_l\}$ be the set of output functions. The output partition, similar to the input partition, can be denoted by (π_1^o, π_2^o). The output partition satisfies the conditions $\pi_1^o \cap \pi_2^o = \phi$ and $\pi_1^o \cup \pi_2^o = \{f_1, f_2, \ldots, f_l\}$. Both processors $P1$ and $P2$ are allowed to compute the output functions belonging to their respective subsets. Before proceeding further, we state some definitions and restate some earlier results.

Since the 1-way communication link is from $P1$ to $P2$, it is not possible to transfer data from $P2$ to $P1$ to compute a function belonging to π_1^o. Hence, an output partition requiring transfer of data from $P2$ to $P1$ is not *feasible*. This leads to the following definition of a *feasible* output partition:

Definition 2 *An output partition is feasible iff on any input partition, all functions in π_1^o can be computed at $P1$ using only the input bits of π_1^i.*

The *1-way information transfer* for multiple output functions can depend on the output partition. Hence, $I_1(F)$, the 1-way complexity of computing a set of

[3] All logarithms in this thesis are to base 2.

output functions is defined as follows:

$$I_1(F) = \begin{array}{c} Min \\ feasible\ output \\ partition \end{array} \left\{ \begin{array}{c} Min \\ input\ partition \end{array} \left\{ \begin{array}{c} worst\ case \\ information\ transfer \\ from\ P1\ to\ P2 \end{array} \right\} \right\}$$

The relationship between I_1 and the minimum area requirement A of a chip stated earlier for single output function holds good for multiple output functions too.

Theorem 1 *The volume V_o of any electro-optical system computing F satisfies $V_o = \Omega(I_1(F))$.*

Proof: Consider an electro-optical system as shown in figure 3.2. $P1$ can be viewed as the electro-optical system and $P2$ as the memory. Consider the state of the system after reading $\frac{n}{2}$ input bits. These bits can be looked upon as bits belonging to π_1^i. Based on this input, the system would have computed some set of output functions. Denote these functions as π_1^o. If the volume of the system is V_o, then the system would not have memorized more than V_o bits of information. It is easy to design a 1-way protocol with V_o bits of information transfer from $P1$ to $P2$ to compute the rest of the output functions. Hence, the system should have at least $I_1(F)$ memory elements. Thus, the volume of the system must be $\Omega(I_1(F))$. □

The computation of F over an input and output partition can be represented in the form of a *computational parallelepiped* P as shown in figure 3.3. $M(N)$ are the set of possible values of the input bits known to $P1(P2)$. For a fixed value of input bits, the output functions in set F are represented by a vector of length l

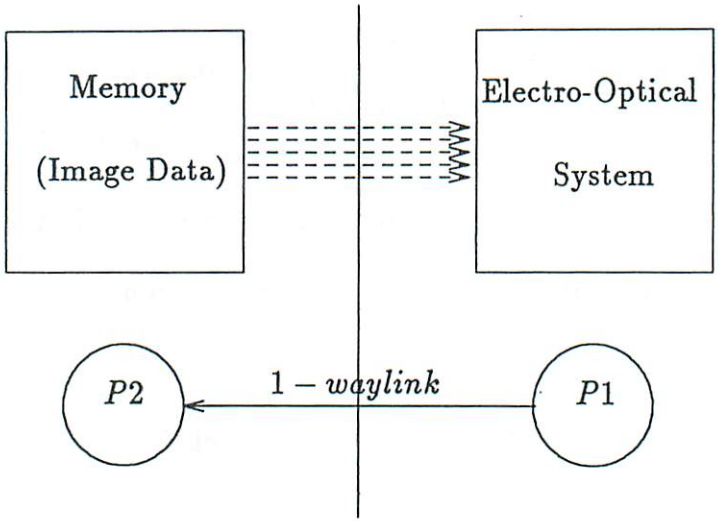


Figure 3.2: 1-way information transfer

in the third dimension of P. Given an input partition, the output functions can be divided into the following subsets:

F_{11} - The subset of the output functions which can be computed at $P1$ (on every input) using the input bits of π_1^i only

F_{22} - The subset of the output functions which can be computed at $P2$ (on every input) using the input bits of π_2^i only

F_{12} - The rest of the output functions.

The computation of the output functions belonging to the subsets F_{12} and F_{22} require data from π_2^i. According to the definition of a *feasible* output partition, these functions can not be computed at $P1$ and, hence, to be computed at $P2$. Therefore, for a given input partition (π_1^i, π_2^i), the output partition is *feasible* iff

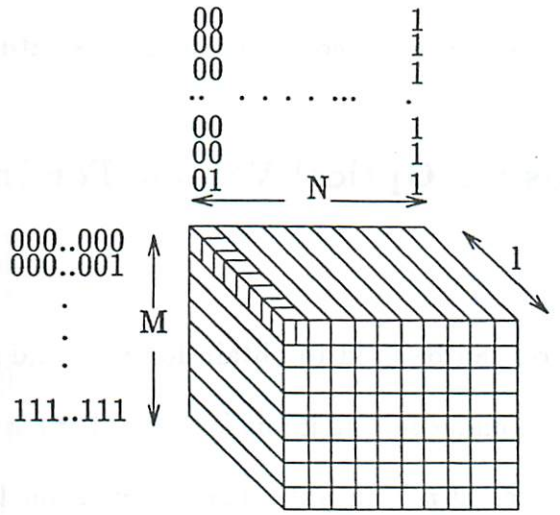


Figure 3.3: The computational parallelepiped P

$F_{12} \subseteq \pi_2^o$ and $F_{22} \subseteq \pi_2^o$.

Since the output functions F_{11} and F_{22} are computed at $P1$ and $P2$ respectively using exclusively the input bits assigned to them, there is no information transfer associated with these computations. The computation of F_{12} only requires information transfer from $P1$ to $P2$. This information transfer $I_1(F)$ is determined from the number of *distinct planes* in P.

Definition 3 *Two planes $i_1, i_2 \in M$ in P are distinct iff there exists a $j \in N$ and a function $f_k \in F$ such that $f_k(i_1, j) \neq f_k(i_2, j)$.*

Based on the above definition of *distinct planes*, $I_1(F)$ can be estimated in a similar spirit as in $I_1(f)$ [125]. This leads to:

Proposition 2 *For a fixed input partition (π_1^i, π_2^i) and a fixed output partition (π_1^o, π_2^o), the minimum number of bits of information transfer from $P1$ to $P2$ to compute F is* $\log d(F)$, *where $d(F)$ is the number of distinct planes in P. Also,*

the 1-way complexity $I_1(F)$ is equal to log d, *where d is the minimum $d(F)$ over all possible input and feasible output partitions.*

In the following section, we use image convolution to illustrate our ideas.

3.1.3 Lower bounds On Optical Volume For Image Convolution

The techniques of last section can be used to obtain lower bound on the optical volume, for image convolution under several input formats. For a $n \times n$ image, the convolution operation produces $O(n^2)$ output. The information transfer $I_1(F)$, for such operation can be analyzed based on the *multiple output function* model discussed above.

Due to the computationally intensive nature of the convolution operation, most of the designs in the literature perform several computations per input pixel fetch to achieve high throughput and reduced memory bandwidth. Figure 3.4 shows the computing environment to carry out convolution using an Electro-optical system.

The input to the system is a $n \times n$ image and a $w \times w$ kernel. The electro-optical system must be able to compute the image convolution on any input $n \times n$ image and any $w \times w$ kernel. The host is responsible for acquiring the input data and feeding it to the system which contains processing units for computation and memory elements for storing intermediate results. Though, most of the practical designs organize cells to input the image pixels from the host in a rasterscan *i.e. scanline* fashion, designs are possible to input data in arbitrary sequence of rows/columns or arbitrary pixel manner.

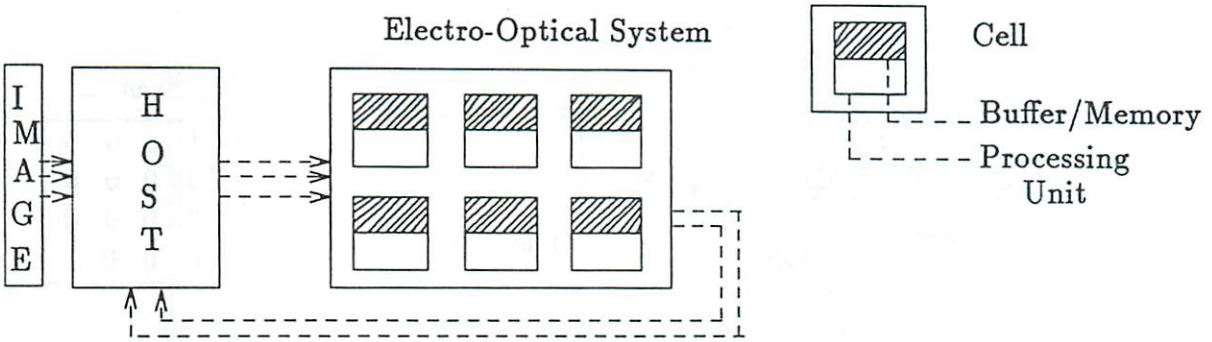


Figure 3.4: Environment for computing image convolution

Scanline Input Case

In this case, the input partition corresponds to $P1$ receiving the upper half of the image and $P2$ receiving the lower half as shown in figure 3.5 (a). Let the kernel K of size $w \times w$ be as shown in figure 3.5 (b). All kernel weights are zero except for $K(1,1)$ and $K(w,w)$ entries. With the given kernel K, the output $C(i,j)$ of the convolution becomes:

$$C(i,j) \;=\; I(i,j) \;+\; I(i+w-1, j+w-1) \quad for \;\; 1 \leq i,j \leq n-w+1.$$

Lemma 1 *[24] The volume V_o of any electro-optical design using scanline input format for image convolution satisfies $V_o = \Omega(nw)$.*

The General Input Case

In this case, we consider designs, where the input is received by the electro-optical system from the host in arbitrary sequence of pixels. We first determine a bound

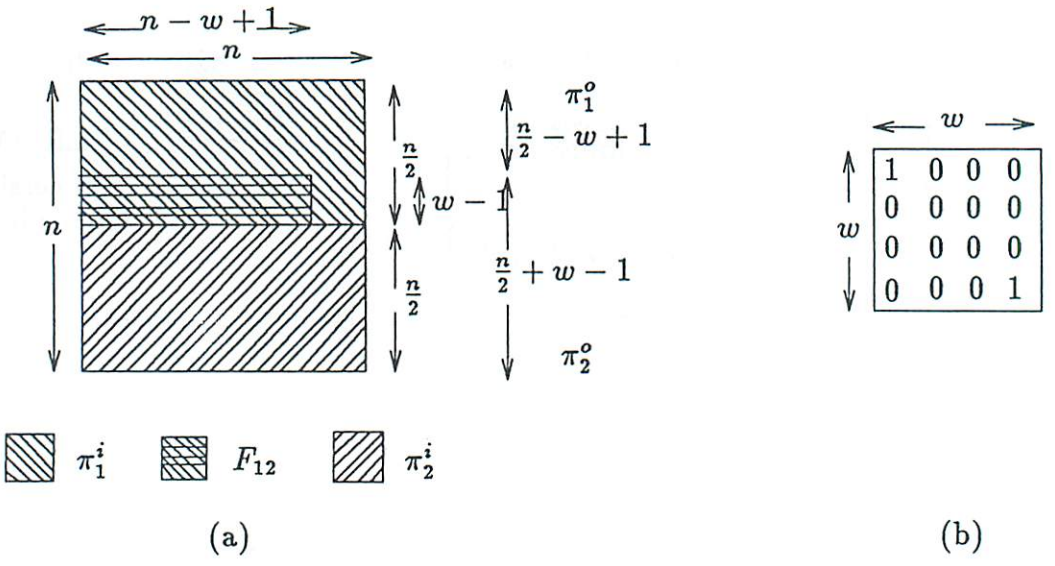


Figure 3.5: (a)The input and output partition for scanline case. (b)A kernel

for a special case of this, where the input is restricted to arbitrary row(column) major sequence. Later we derive the bound for the general case by reducing it to the arbitrary row(column) major input format.

Theorem 2 *[24] The volume V_o of any electro-optical design for convolving a $w \times w$ kernel with a $n \times n$ image satisfies $V_o = \Omega(nw)$.*

3.2 Upper bounds

In this section, we present a class of optical interconnection networks as a realization of the OMC presented in the previous section. Each of the proposed designs uses a different optical device technology for redirection of the optical beams to establish a new topology at any clock cycle, and represents an upper bound on the volume requirement of OMC.

3.2.1 Optical Mesh Using Mirrors

In this design, there are N processors on the processing layer of area N. Similarly, the deflecting layer has area N and holds N mirrors. These layers are aligned so that each of the mirrors is located directly above its associated processor (see Figure 3.6). Each processor has two lasers. One of these is directed up towards the arithmetic unit of the mirror and the other is directed towards the mirror's surface. A connection phase would consist of two cycles. In the first cycle, each processor sends the address of its desired destination processor to the arithmetic unit of its associated mirror using its dedicated laser. The arithmetic unit of the mirror computes a rotation degree such that both the origin and destination processors have equal angle with the line perpendicular to the surface of the mirror in the plane formed by the mirror, the source processor, and the destination processor. Once the angle is computed, the mirror is rotated to point to the desired destination. In the second cycle, connection is established by the laser beam carrying the data from the source to the mirror and from the mirror being reflected towards the destination. Since the connection is done through a mechanical movement of the mirror, with the current technology this leads to an order of milli-second reconfiguration time. Therefore this architecture is suitable for applications where the interconnection topology does not have to be changed frequently. In [70], the design of various topologies have been studied to minimize the time complexity of several problems for fixed period of computation.

The space requirement of this architecture is $O(N)$ under the following assumption. Each mirror is attached to a simple electro-mechanical device which

takes one unit of space and can rotate to any position in one unit of time. With current technological advancements, the above assumption may not be the most accurate one but we do not see any technological limitations preventing us from making our assumptions. In fact, our assumptions are as valid as those in VLSI; the constant propagation delay assumption regardless of wire's length. Other assumptions can also be made based on the following arguments. Many mirrors have a reconfiguration delay proportional to their rotation angle, $O(N)$. More complex mirrors on the other hand, can rotate faster for a larger angle (unit time rotation delay) but their size can grow proportional to the number of angles they can realize ($O(N)$).

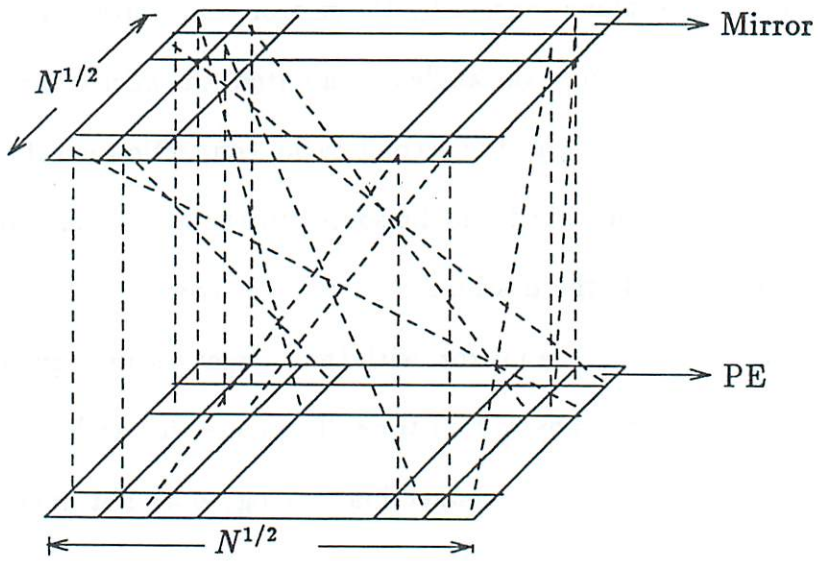


Figure 3.6: An Optical mesh using mirrors

3.2.2 Reconfiguration Using Acousto-Optic Devices

In this organization, N processors are arranged to form a one-dimensional processing layer and the corresponding acousto-optics devices are similarly located on a one-dimensional deflecting layer (see Figure 3.7). The size of each of the acousto-optic devices is proportional to the size of the processing array, leading to an $O(N^2)$ area deflection layer. Similar to the design using the mirrors, every processor has two lasers, and each connection phase is made up of two cycles. In the first cycle, each processor sends the address of its desired destination processor to the arithmetic unit of its associated acousto-optic unit using its dedicated laser beam. The acousto-optic cell's arithmetic unit computes the frequency of the wave to be applied to the crystal for redirection of the incoming optical beam to the destination processor. The acousto-optic device then redirects the incident beam from the source to the destination processor. One of the advantages of this architecture over the previous design is its order of micro-seconds reconfiguration time, which is dominated by the speed of sound waves. The other advantage is its broadcasting capability, which is due to the possibility of generating multiple waves through a crystal at a given time. Furthermore, the above can be extended to interconnect a two dimensional grid of processors as follows.

Proposition 3 *Using a* $(N^{1/2} \times N^{1/2})$ *processing layer, and* $(N^{1/2} \times N^{1/2})$ *array of acousto-optic devices as the deflecting layer, one step of OMC can be realized in* $O(\log^2 N)$ *time and* $O(N^2)$ *area.*

The area is obtained with similar arguments as in the one dimensional case. The time complexity is due to the movement of data using the standard divide and conquer techniques. At the i^{th} step a block size 2^i is divided into two blocks

of half the size. Each subblock only contains the data elements destined to its memory locations. To route up $O(i)$ elements residing in the queue of each of the processors, the i^{th} step is simulated by $O(i)$ iterations.

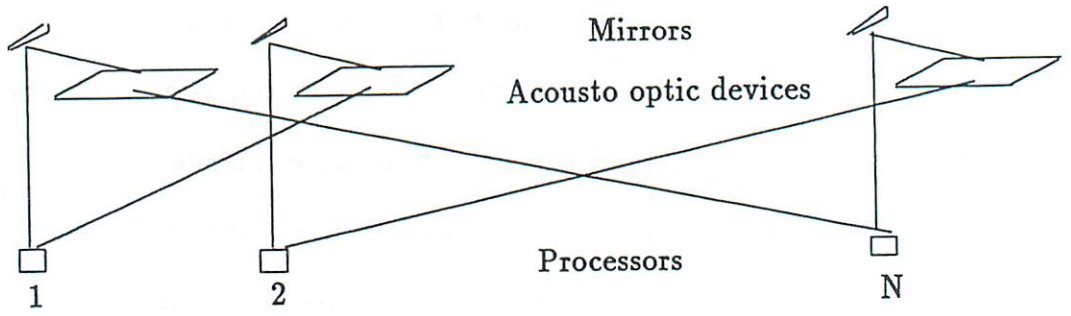


Figure 3.7: Reconfiguration using acousto-optic Devices

3.2.3 Electro Optical Crossbar

This design uses a hybrid reconfiguration technique for interconnecting processors. There are N processors each located in a distinct row and column of the $N \times N$ processing layer. For each processor, there is a hologram module having N units, such that the i^{th} unit has a grating plate with a frequency leading to a deflection angle corresponding to the processor located at the grid point (i, i). In addition, each unit has a simple controller, and a laser beam. To establish or reconfigure to a new connection pattern, each processor broadcasts the address of the desired destination processor to the controller of each of N units of its hologram module using an electrical bus (see Figure 3.8). The controller activates a laser (for conversion of the electrical input to optical signal), if its ID matches

the broadcast address of the destination processor. The connection is made when the laser beams are passed through the predefined gratings. Therefore, since the grating angles are predefined, the reconfiguration time of this design is bounded by the laser switching time which is in the order of nano-seconds using Gallium Arsenide (GaAs) technology [56].

This architecture is faster than the previous designs and further it compares well with the clock cycle of the current supercomputers. One of the advantages of this simple design is in its implementability in VLSI, using GaAs technology. Unlike the previous designs, this can be fabricated with very low cost and is highly suitable for applications where full connectivity is required. In such applications, the processor layer area can be fully utilized by placing N optical beam receivers in each of the vacant areas to simultaneously interconnect with all the other processors. This design can be easily adopted to implement a neural network of processes with optical interconnects [28].

3.3 Performance Issues

In this section, we discuss the performance characteristics of the architectures proposed in section 3.2. We concentrate on five issues of connectivity, size, speed, power and energy, and cost. This order of presentation reflects its relevance to the focus of this thesis.

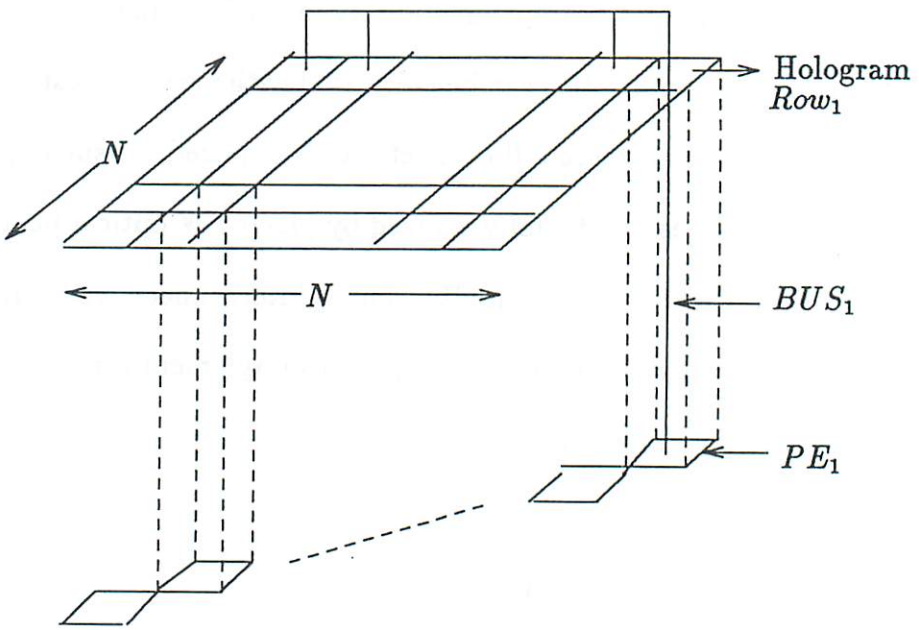


Figure 3.8: Reconfiguration using an electro optical crossbar

3.3.1 Connectivity

The optical mesh, optical linear array, and electro optical crossbar can realize any permutation in unit time. However, each differs in its broadcasting capability. Unlike a mirror, an acousto optic device can be used as a deflector with broadcasting capability. There is no theoretical limit on the maximum number of acoustic waves that can be propagated simultaneously. But due to the resolution offered by the state of art technology is in the order of few thousands [101], this is limited to a constant fan out. The electro-optical crossbar on the other hand can provide full broadcasting capability. We also showed that even the weaker modified versions of these models lead to efficient solutions to meet the connectivity needs. As an example a 2-dimensional array with AO devices was shown to simulate the its 1-dimensional version in $O(\log^2 N)$ time. Other possible variations are also interesting to be studied. This study should include the proposal of more practical designs reflecting the state of art technology.

3.3.2 Size

To compare the sizes of these devices, we have to consider several factors such as the number of devices, size of each device, functionality of each device, and the total space occupied by their arrangements. The optical mesh and optical linear array use linear number of deflectors while the electro optical crossbar uses linear number of deflecting units each having linear number of cells. On the other hand, the size of each AO deflectors is linear to the number of processors. Hence, based on the assumptions we made earlier for optical mesh, its space requirement is

superior to both the optical linear array and electro optical crossbar by a linear factor. It is also not right to evaluate the space requirement of the optical linear array and electro optical crossbar to be equivalent, since one is a one dimensional organization and the other falls under a higher dimension. From a practical point of view however, the results are exactly opposite of above; electro optical being the smallest, and optical mesh the largest. The space requirement of the electro optical mesh is dominated by the area requirement of mirrors. Integrated GaAs laser diodes are available at the size of 30 microns each [56]. This is smaller than a mirror or an AO device by a factor of one thousand, and can be well used at the gate level. The size of a mirror and an AO device is compatible with a powerful VLSI processor so can be used for interprocessor communication.

3.3.3 Speed

Since the gratings are predefined in the electro optical crossbar, the reconfiguration delay is dominated by the lasers' switching time. Integrated GaAs diodes are available with the switching time in an order of nanoseconds. The reconfiguration time using AO devices is limited by the speed of sound and leads to an order of micro-second using current size devices. However, when used as a systolic array to resolve 1000 input/outputs at a time, can lead to nano-second reconfiguration time, as the size of input scream goes to ∞. The slowest organization is the optical mesh using mirrors. Using mechanical mirrors, its reconfiguration time is in an order of milli-seconds. For such an organization, it is desired to not change the configurations often. For example, in computing FFT the connections can be set as a Shuffle exchange topology. In [70], they have studied to design topologies

for various problems in order to minimize the communication overhead and the need for reconfiguration.

3.3.4 Power and Energy

Replacing each electrical line with an optical equivalent would cause an increase in complexity and system power. For a digital signal, a single optical interconnect requires a voltage to current converter to drive the light emitting diode or laser, optical coupling mechanisms to direct optical energy through free space, a detector, preamplifier, filter, and thresholding circuit, and possibly an automatic gain control circuit. Compared with an electrical interconnect, the complexity of an optical interconnect is obviously larger, and the power is comparable for an LED-driven interconnect or larger for a laser driven interconnect [48]. The typical threshold current of the laser is 20 to 30 mA, the power consumption when on is about 0.2 W, and the expected thermal power dissipation is about 0.2 W per millimeter square for lasers separated by 1 millimeter [12]. For the implementation of electro optical crossbar in a GaAs circuitry (enhancement mode), the power dissipated in a single gate is approximately 0.2 mW gate at 500 MHz, with power (gate size) scaling approximately with required frequency.

3.3.5 Cost

At this early stage, it is difficult to predict the cost of the optical computer. But with the larger powers of some of the devices, one cost that seems sure to dominate is that of the laser power supply. Today's large supercomputers use

similar levels of power as an optical computer. But its high-quality electronic power supply costs about one to two dollars per watt, while lasers cost thousands of dollars per watt of high quality output [121]. Even the diode laser is between $1000 and $10,000 per watt, if its price is divided by its small power output. Because cost at this level involves numerous factors, many not related to actual construction of the computer, such as marketing issues, and volume of production, it is impossible to determine a price.

3.3.6 Fault Tolerance

The proposed optical architectures are very suitable fault tolerant designs, since the connections can be reconfigured to operate with out the faulty processors or deflectors. In case of a processor failure, its data has to be distributed among other processors. These processors then reconfigure their connections to meet the need of distributed data. However in case of a deflector failure a processor needs to establish its desired connection through its nearest non-faulty processor. As shown in Figure 3.9, the processor i accesses its nearest nonfaulty processor $i+x$, and uses this deflector to connect to destination j. Where x is the Euclidean distance between i and $i+x$. The sequence of events is shown as A, B, C.

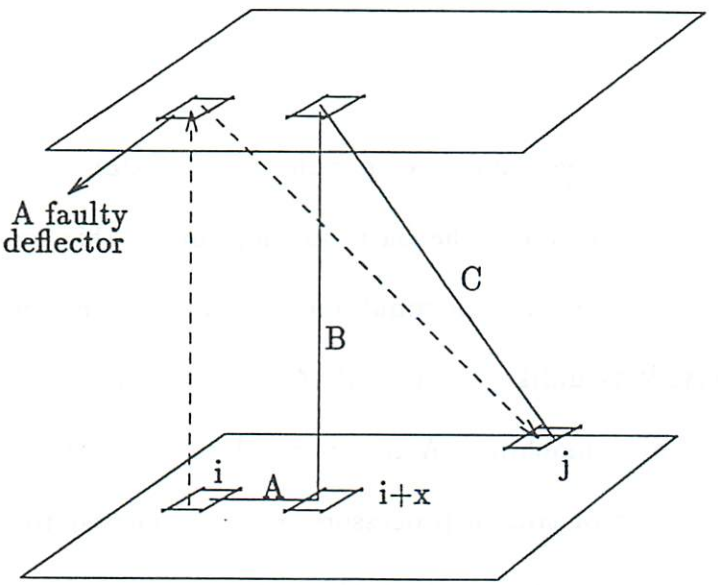


Figure 3.9: A Fault tolerant scheme

Chapter 4

Simulation of Shared memory

One of the widely used models of parallel computation is the Parallel Random Access Machine (PRAM). The basic assumption in this model is that in unit time each of N processors can simultaneously access one memory location [124]. Unfortunately, it is unlikely that a PRAM model will ever faithfully represent any "real" parallel machine. A real parallel computer will most likely consist of a large number of simple processors, each connected to a small number of other processors. Each processor in this network has its own local memory, and processors communicate by sending messages over links to neighboring processors. To reconcile the convenience of a PRAM with the limitations of a real computer, it is simulated on a real network.

One of the first randomized simulations is given in [118], where it is shown that there exists an N-processor realistic computer that can simulate an idealistic N processor parallel computer with only a factor of $O(\log N)$ loss of run time efficiency. In [93], this was improved by obtaining similar bounds but requiring

only bounded queue size.

Recently, there has been an emerging interest in the design of models of parallel computation which more closely simulates a realistic machine. In [45], a more restricted PRAM model called Distributed Random Access Machine (DRAM) is introduced, which reflects an assumption of limited communication bandwidth in the underlying network. All memory in DRAM is local to the processors, and is accessed by routing messages through a communication network. A stronger model called local memory PRAM was introduced in [4]. Like the DRAM, the memory is distributed. However, there is no restriction on the underlying communication network and hence it is assumed to have a unit time delay. Such a communication medium is feasible with fixed connection architectures of unbounded degree, or those with reconfigurable optical interconnects.

In this chapter, we present simple efficient algorithms for simulation of a N processor EREW PRAM using an OMC with $N/\log N$ processors. Section 4.1 presents the algorithm with its deterministic analysis, while in section 4.2 the running time of its randomized version is shown to lead to an sub-optimal solution. The input to these simulation algorithms is assumed to have the following form. Every processor is responsible for routing the $(\log N)$ messages that reside in its local memory. Each message has a destination tag attached to it. The destination address is made up of two components x, y, where x denotes the address of the memory module, $x \leq N/\log N$, and y denotes its position within the module, $y \leq \log N$.

4.1 Deterministic Simulation

An OMC with N processors can simulate, in real time, an Exclusive Read Exclusive Write PRAM having P processors and M memory locations, where $N=$ maximum $\{P, M\}$. On the other hand, a P processor EREW PRAM can simulate in real time the computations of a P processor OMC. Thus, the interesting cases are when the memory is "large". In the following, the number of processors in the OMC is less than the number of processors in the PRAM by a factor of $\log N$.

Lemma 2 *One step of an N processor EREW PRAM can be simulated on an OMC with $N/\log N$ processors in $\Theta(\log^2 N)$ time. Further, there exists an input sequence for which this is the best possible bound.*

Proof: In this algorithm, all elements of each processing element (PE) are sorted based upon the position to which they will write within the memory module, with duplicate positions being sent to a second queue. This prevents more than one processor trying to write to the same module of memory, because there is only one of each of the positions within each module. Next, each of the elements is sent out one by one. After this the duplicates from the last iteration are brought forward and the process is repeated. This algorithm works in $O(\log^2 N)$ time by running through each $O(\log N)$ time iteration a total of $\log N$ times, to insure that all elements are transferred to the correct memory locations. A sequence for which this is the best possible bound is shown in Figure 4.1. The following is the outline of the algorithm. The complete code can be found in Appendix I. □

```
for x2 = 1 to log N ; each iteration is done log N times
    call sort ; sorts by memory position in each module
    call send ; sends the elements to their correct locations
```

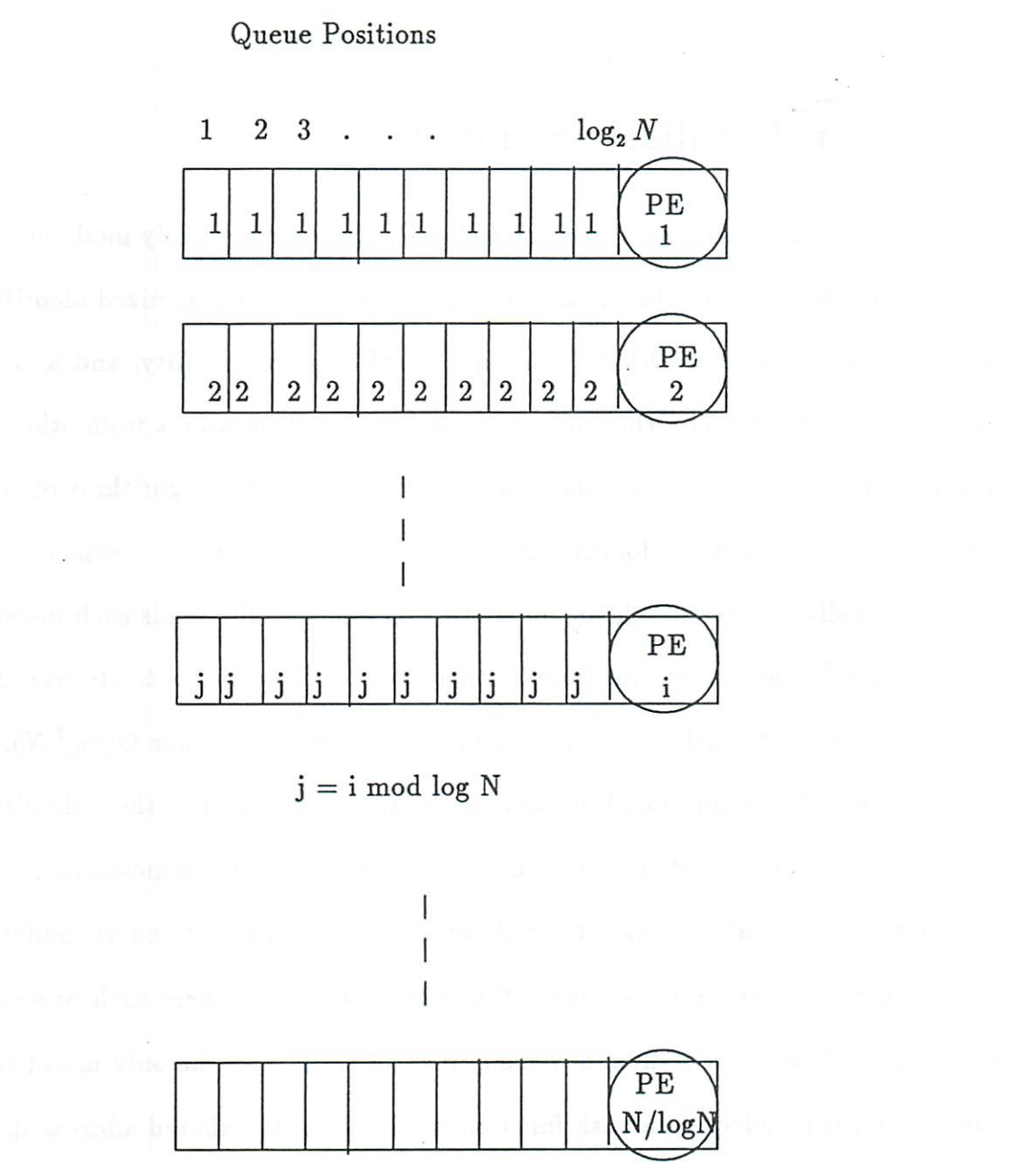


Figure 4.1: Worst input sequence

```
    call q2-q1 ; the duplicates are brought forward
next x2 ; another iteration
end ; end of complete process
```

4.2 Probabilistic Simulation

The following analysis shows that the $O(\log N)$ stages of a slightly modified version of the algorithm in the proof of Lemma 2, leads to a randomized algorithm with a running time of $O(\log N \log\log N)$ with high probability, and a worst case time complexity of $O(\log^2 N)$. Our routing algorithm differs from others in the literature in the way randomization is used. Unlike the algorithms of [118] and [116], it does not randomize with respect to paths taken by messages. For example, Valiant's classic scheme for routing on a hypercube sends each message to a randomly chosen intermediate destination and, from there, to its true destination. On OMC such a technique would lead to queues of size $O(\log^2 N)$. In [45], instead of choosing random paths for messages to traverse, their algorithm repeatedly attempts to deliver a randomly chosen subset of the messages. A by-product of their strategy is that their algorithm requires no intermediate buffering of messages, and hence works under the general situation where each processor can send and receive polynomially many messages. In [93], the only use of randomization is in selecting a hash function to distribute the shared address space of the PRAM onto the nodes of the butterfly. (Note that the direct application of the techniques of [93] to our problem will lead to $O(\log^2 N)$ running time, and $O(\log^2 N)$ local memory requirement for each processor.) Similarly, we only use randomization in assuming a random distribution for input data. This random

distribution can also be obtained in selecting a hash function to distribute the shared address space of the PRAM onto an OMC with $N/\log N$ processors, each having $\log N$ local memory. The routing itself is deterministic as explained in the proof of Lemma 2, and has worst case running time of $O(\log^2 N)$.

Theorem 3 *The probability that the simulation of one step of an N processor EREW PRAM using an OMC with $N/\log N$ processors would take more than $O(\log N)$ time is given by $\beta e^{-\log^2 N\beta}$, for some constant β independent of N.*

In this algorithm, as opposed to the first one, we check each iteration for how many elements are left to be sent, and if there are none, the algorithm exits. If there is still work to be done, however, the program will run again, but only go up to the number of elements left to be sent. This leads to $O(\log N \log\log N)$ running time with a high probability. The following code summarizes the algorithm. For complete code see Appendix I.

```
call init ; this initializes the pointers
do forever ; it loops forever until done
    l = log N ; the looping variable
    call sort2 ; sorts by memory position using pointer jumps
    call send2 ; send out the elements using pointer jumps
    if p = 1 then end ; if done then exit
    call q2-q12 ; switch queues with pointer jumps
    call point ; adds new pointer jumps as certain elements are
                 completely finished
    l = p ; renew looping variable
end ; do loop again
```

Proof: To show that the above algorithm with a high probability has a running time of $O(\log N \log\log N)$, we have to show that at least one half of the messages will be routed after each stage, with a high probability. We do this by showing that at least one half of the messages at each module are distinct, at the beginning

of each stage. To begin the algorithm, there are $n = \log N$ messages in each queue, and we are interested in having at most $k = \log N/2$ identical labels (duplicates), where the probability of having a given label is $p = \log N/N$. The probability of having exactly k identical labels can be estimated by using Bernoulli trials, $b(k, n, p)$. The probability of having more than k identical labels can be approximated by using the Poisson distribution as follows [33]:

$$P[S_n \geq k] \leq e^{-\lambda}\lambda^k/k!(kq/k - np)$$

where $q = 1 - p$. This can be approximated to

$$\beta e^{-\log^2 N\beta}$$

for some constant β, independent of N. Similar analysis can be applied to the rest of the stages of the algorithm, where the queue size reduces by half each time. The stated overall probability is an upper bound obtained by the product of the probabilities from each of the $\log N$ stages. □

We have also verified the above results by computer simulations. This is illustrated in Figure 4.2.

The above loosely synchronized point based techniques can be used as general subroutines for solving many problems on the OMC. An immediate consequence of our technique is to list ranking. The list ranking problem is: given a linked list in memory, compute the distance that each cell is from the end of the list. The problem is a fundamental data structure problem, and has many applications. There are many problems that are reducible to list ranking such as, parallel tree contraction [73, 39].

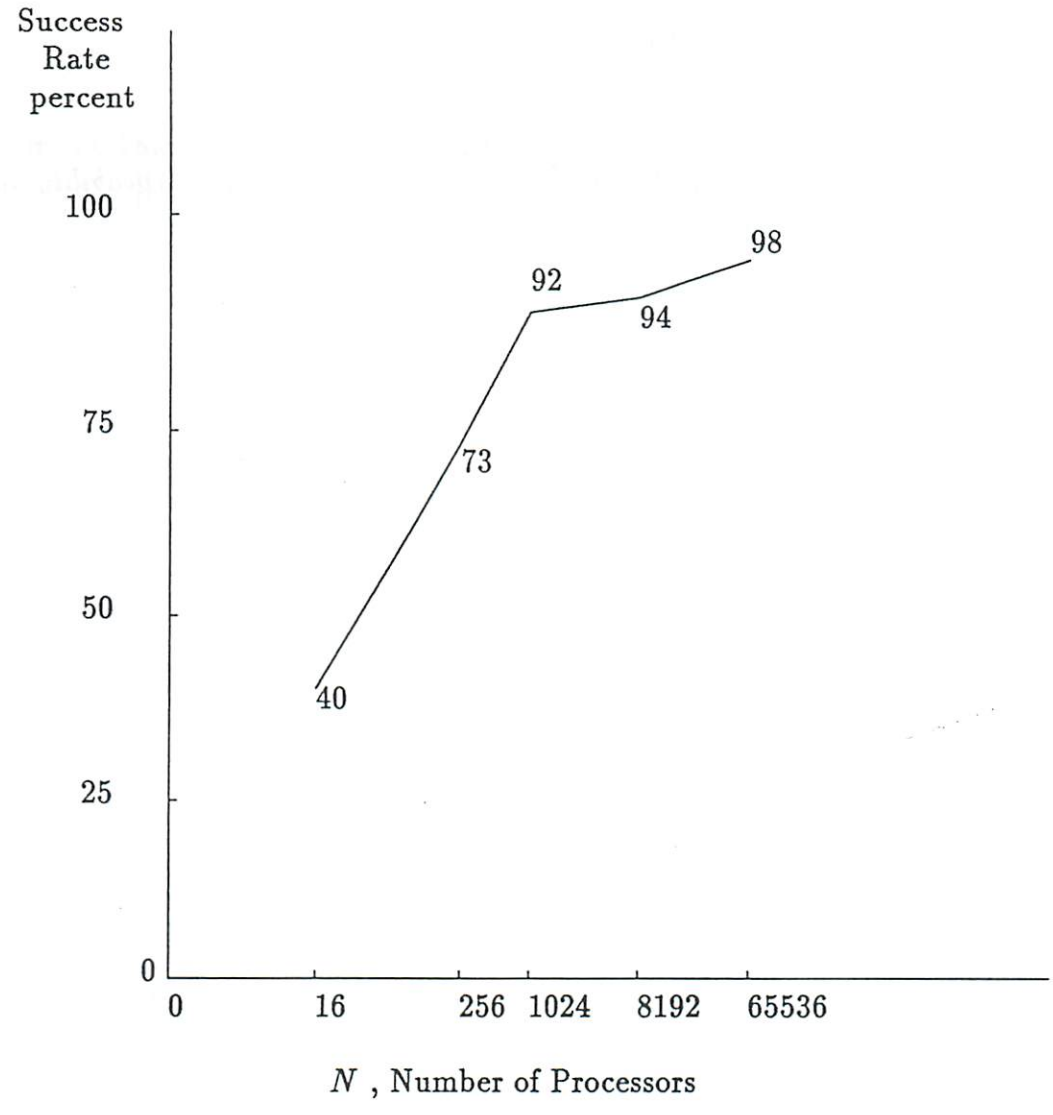


Figure 4.2: Simulation results

The list ranking, problem has been studied extensively with the goal of getting $N/\log N$ procesor, $O(\log N)$ time PRAM algorithms [21]. In [4], it is shown that these bounds can be met on the local memory PRAM for a randomized algorithm. Compared to their algorithm, our solution is more attractive since it guarantees a worst case running time of $O(\log^2 N)$, as stated below.

Corollary 1 *The list ranking problem of size N can be performed on an OMC with $N/\log N$ processors, in $O(\log N \log\log N)$ time with high probability and in $O(\log^2 N)$ time otherwise.*

Part III

Applications

Chapter 5

Signal and Image Processing

Recently, there has been an emerging interest in the design of parallel algorithms that are efficient in solving image problems [23, 76, 84, 96, 98]. The performance of these algorithms is very much dependent on the underlying parallel architecture. Several mesh based architectures implementable in VLSI have been considered for fine grain image computations, where each pixel is directly mapped onto a processing element. Solving many image problems on a $N \times N$ mesh array of processors takes as much as $O(N)$ time [76]. To provide faster solutions, several other architectures such as the pyramid, the mesh of trees, and the reconfigurable mesh have also been considered [74, 84, 86]. Their hierarchical organization leads to logarithmic time performance in solving some problems. But for solving more communication intensive problems, they are not superior to a two dimensional mesh.

The PRAM has also been considered for designing parallel algorithms for problems in computational geometry and computer vision. Simulation of this

model on the OMC was shown in the previous chapter. In this chapter, we consider fine grain electro-optical arrays for optimal parallel solutions to several problems in signal and image processing. We illustrate the use of OMC in fine grain image computations for problems such as labeling figures in a image, determining the convex hull of all figures, and finding the nearest neighboring figure to each figure. We also show how iterative methods for a class of problems in image understanding can be performed efficiently on the OMC with holographic interconnects.

5.1 Optical Mesh and EREW Algorithms

In the previous chapter, we studied the relationships between OMC and the shared memory models. It is easy to see that an OMC with N processors can simulate, in real time, an Exclusive Read Exclusive Write PRAM having P processors and M memory locations, where $N = \text{maximum } \{P, M\}$. In this section, we formally define an optical mesh and illustrate its operation in simulating EREW algorithms.

Definition 4 *An optical mesh of size N^2 has a processor layer consisting of a $N \times N$ array of processors which can intercommunicate in unit-time using their corresponding optical device residing on the deflection layer of same size.*

A simple implementation of optical mesh is possible using mirrors (under the unit delay rotation assumption) which was discussed in the previous chapters. The communication patterns needed for performing the following computations can be easily realized using the optical mesh:

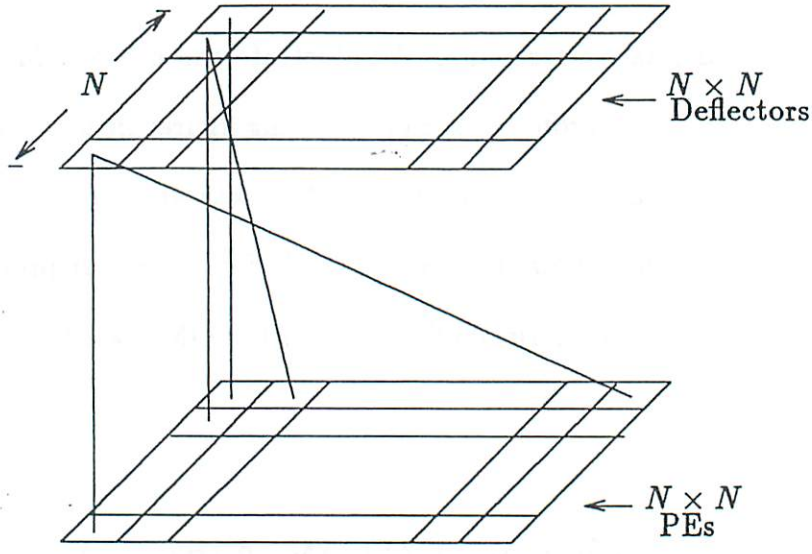


Figure 5.1: Matrix transpose

1. The transpose of a $N \times N$ matrix can be computed in $O(1)$ time using an optical mesh of size $N \times N$ (see Figure 5.1).
2. Two $N \times N$ matrices can be multiplied in $O(\log N)$ time using $O(N^3/\log N)$ processors, using an $(N^2/\log N) \times N$ optical mesh.
3. The polynomial evaluation of degree N can be performed in $O(\log N)$ time, using $(N/\log N) \times 1$ optical mesh.
4. The FFT of N points can be computed in $O(\log N)$ time, using N processors on the $N \times 1$ optical mesh (see Figure 5.2).
5. N elements can be sorted in $O(\log^2 N)$ time, using N processors on the $N \times 1$ optical mesh.

To familiarize the reader with the above model and some of the basic parallel techniques used in designing algorithms in this section, we describe the following image template matching algorithm. Template matching is a basic operation in image processing and computer vision [98]. It is used as a simple method for filtering, edge detection, image registration and object detection [105]. Template

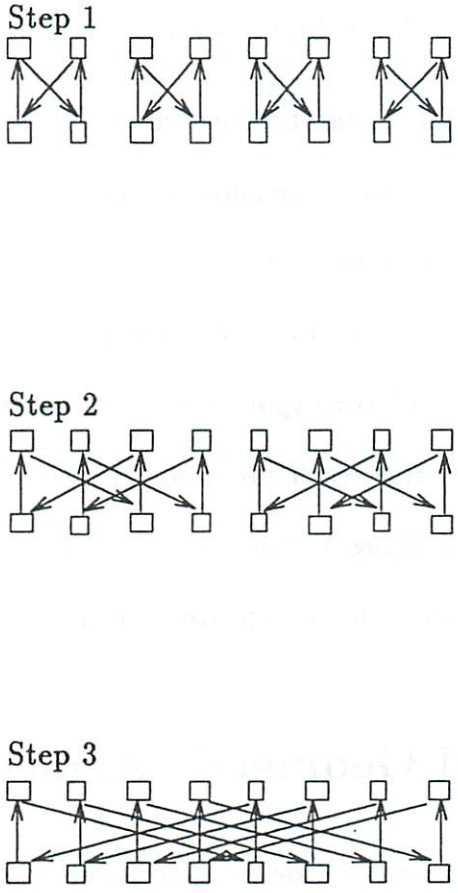


Figure 5.2: Communication pattern for FFT computation

matching can be described as comparing a template (window) with all possible windows of the image. Each position of the image will store the result of the window operation for which it is the top-left corner. Let $IMAGE\ (i,j)$ represent an $N^{1/2} \times N^{1/2}$ image where $0 \leq i,j \leq N^{1/2} - 1$. Let $W(s,t)$ represent the template where $0 \leq s,t \leq k-1$. Then, the result $C(i,j)$, is as given below:

$$C(i,j) = \sum_{s=0}^{k-1} \sum_{t=0}^{k-1} IMAGE((i+s) mod N^{1/2}, (j+t) mod N^{1/2}) * W(s,t)$$

Note that the computation can be done in $O(N \times k^2)$ time using a uniprocessor.

Using $N/\log N$ processors, the above computation can be done in $O(k^2 \log N)$ time. This is simply done by first partioning the processors into groups of size $(k/\log^{1/2} N) \times (k/\log^{1/2} N)$. Each of these groups checks the template of size $k \times k$ with the image of corresponding size. Since the number of processors is less than the image size, each processor is assigned to check a region of size $\log^{1/2} N \times \log^{1/2} N$. In $O(\log N)$ time, the results are obtained for this particular position of the template. This is repeated for all k^2 positions.

5.2 Optimal Geometric Algorithms

In this section, we present $O(\log N)$ algorithms for problems such as finding connected components, determining the convex hull of all figures, and nearest neighboring figure to each figure in a $N \times N$ image. The input to our algorithms is a *digitized* picture with $PE(i,j)$ storing the pixel (i,j), $0 \leq i,j, \leq N-1$ in the plane.

We are concerned with black and white (binary) images, where the black pixels are 1-valued, and white pixels are 0-valued. Connectivity among pixels can be

defined in terms of their adjacency. Two black pixels (i_1, j_1) and (i_2, j_2) are *8-neighbors* if $\max\{|i_1 - i_2|, |j_1 - j_2|\} \leq 1$, and *4-neighbors* if $|i_1 - i_2| + |j_1 - j_2| \leq 1$. Two black pixels (i_1, j_1) and (i_k, j_k) are said to be connected by a *8-path(4-path)* if there exists a sequence of black pixels (i_p, j_p), $2 \leq p \leq k$, such that each pair of pixels (i_{p-1}, j_{p-1}) and (i_p, j_p) are 8-neighbors(4-neighbors). A maximal connected region of black pixels is called a *connected component*.

In a 0/1 picture the connected 1's are said to form a figure. Thus, associated with each PE is a label which is the unique ID of the figure to which the PE belongs. The label associated with PE's (i, j) and (r, s) are the same iff (i, j) and (r, s) are connected by a series of 1's. For more details see [76].

Given a digitized 0/1 picture with PE(i, j) storing the pixel (i, j), the convex hull of the 1's is the smallest convex polygon enclosing the 1's. An algorithm to mark the extreme points will store a value say 1 if the PE is an extreme point, 0 otherwise. Enumeration of extreme points corresponds to marking as well as numbering the extreme points in some order, such as in clockwise order starting from the top rightmost extreme point.

5.2.1 Labeling Digitized Images

An early step in image processing is identifying figures in the image. Figures correspond to connected 1's in the image. An $N \times N$ digitized picture may contain more than one connected region of black pixels. The problem is to identify to which figure (label) each "1" belongs to.

Lemma 3 *Given a $N \times N$ 0/1 image, all figures can be labeled in $O(\log N)$ time using an $(N \times N)$-optical mesh.*

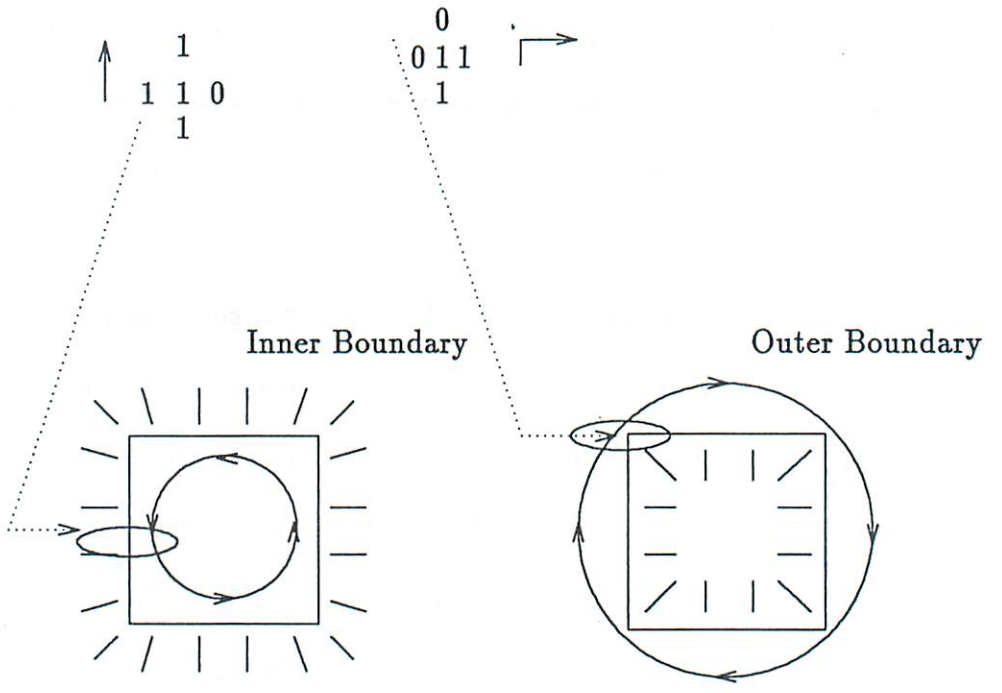


Figure 5.3: Boundary rules

Proof: The basic idea of the algorithm is to identify the outer and inner boundaries of each figure, and then uniquely label all the connected ones surrounded by each of these boundaries [2]. To assure circular boundaries, the input image is magnified by a factor of two, along each dimension. Each pixel then, locally determines whether it is a boundary pixel or not by checking if at least one of its four adjacent pixels along the x and y axis, hold a "0". The pixels along each boundary is linked to form a circular list. The direction of pointers are determined as shown in Figure 5.3. The details for two selected segments are shown. Others can similarly be formulated.

Now on, only the boundary PEs take part in the computation to identify the

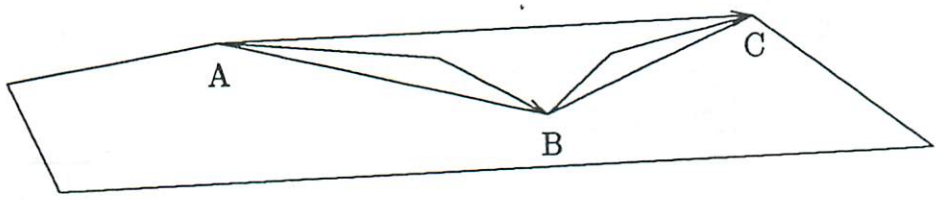


Figure 5.4: Pointer jumping

least numbered PE in their list. Each PE during iteration $i+1$, sets its pointer to the pointer of the PE it was pointing to at the end of iteration i. In Figure 5.4, the i^{th} iteration for some i, has A pointing to B and B pointing to C. In the next iteration, A points to the PE that B was pointing to, which is C. Since this has the effect of doubling the distance "jumped" during each iteration, in $O(\log N)$ time all the PEs in each list know the least numbered PE in their list. The final step is the propagation of the unique IDs of each of outer boundaries to its inner region. Broadcasting of IDs in done in parallel along each row of the image. It is easy to see that since the figures do not cross there is always a unique ID broadcasted to each of the inner PEs. □

The following code is the skeleton of the algorithm which each of the processors could run simultaneously to label the figures in an $N \times N$ image in $O(\log N)$ time. The complete code can be found in Appendix II.

```
procedure Label ; main program
    call Multiply ; multiplies picture fourfold
    call Border ; each border point points to its neighbor
    call Unify ; all borders are labeled uniquely
    call Propagate ; all outer borders propagate their labels
end
```

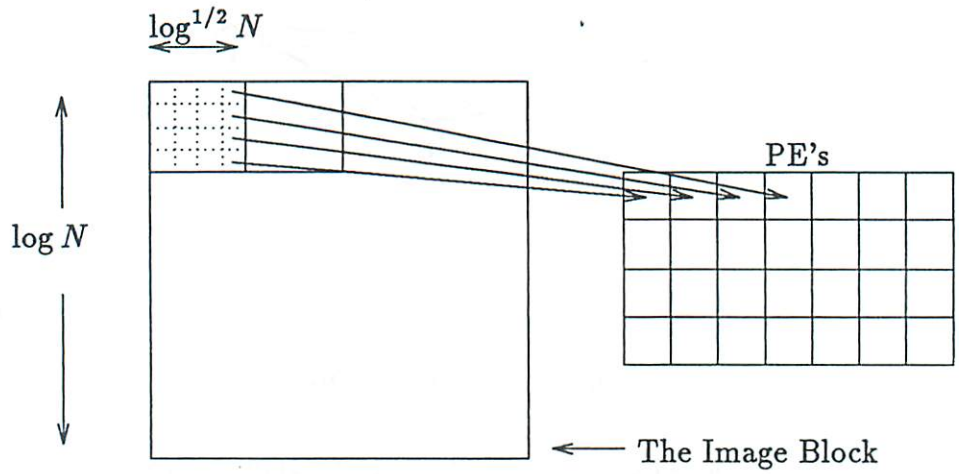


Figure 5.5: Merging of blocks and processor assignment

In the following, we use less processors to lead to optimal solution.

Theorem 4 *Given a $N \times N$ 0/1 image, all figures can be labeled in $O(\log N)$ time using an $(N/\log^{1/2} N \times N/\log^{1/2} N)$-optical mesh.*

Proof: In first step, we assign a $\log^{1/2} N \times \log^{1/2} N$ block of image to each processor, and sequentially label the figures within these regions. This is accomplished using a serial graph traversal technique (see Appendix III):

```
procedure Main.Label ; main program
    call Border ; determines internal borders
    from loop=1 to c^2; c^2 is the number of pixels in each PE
        call Edge ; check for PE boundary
        if true then call Unify ; if so, then unify boundary
    continue loop ; again, until done
end
```

In the second step, these blocks are merged together until the block size becomes $\log N \times \log N$ (see Figure 5.5). During each iteration, four blocks of size $k \times k$ are merged to obtain a block of size $2k \times 2k$. This is performed by assigning the available PEs to block boundary pixels, and then applying the algorithm of lemma 3 to merge each pair of blocks. Since there are not enough processors available to hold all the block boundary points, they are processed by groups of $\log N$ at a time. Hence, the total time to simulate each of $\log \log N$ iterations is $O(\log^{1/2} N)$. This leads to a total of $O(\log^{1/2} N \log \log N)$ time complexity for the second reduction step. Using Lemma 3, the remaining pixels are labeled.

5.2.2 Convexity Algorithms

Convexity plays an important role in image processing and in vision; many other problems can be solved once the convex hull of figures is obtained. It has applications to normalizing pattern in image processing, obtaining triangulations of sets of points, topological feature extraction, shape decomposition in pattern recognition, testing for linear separability, etc. [75].

Theorem 5 *Given a $N \times N$ 0/1 image, the convex hull of all figures can be enumerated in $O(\log N)$ time using an $(N \times N)$-optical mesh.*

Proof: First, identify the figures and make a list of pixels in the outer boundary of each figure. This can be done using the algorithm described for the labelling problem. To begin with, each boundary pixel is defined to be an extreme point. In this algorithm each boundary pixel checks the largest enclosing angle made in the region between the two boundary pixels which it is pointing to on either side of it. If this angle ≥ 180 degrees then it eliminates itself from the list of extreme

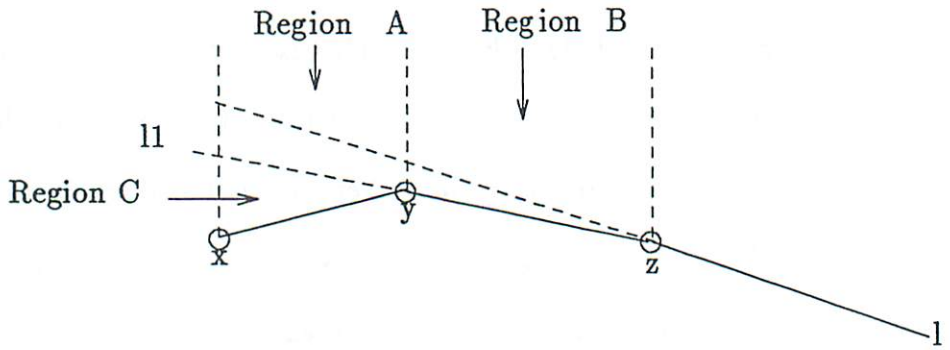


Figure 5.6: Proof of correctness

points. If not, it continues along the boundary, doubling the distance covered during each iteration, to compute the angle with other boundary pixels. This is repeated over $O(\log N)$ iterations (Details are given in Appendix II).

```
procedure Convex.hull ; main program
    call Check ; initialization of variables, all
                 non-boundary processors are eliminated
    while l < log E do ; l=loop counter; E=size of image (N)
    call Angle ; computes the angle between processors
    call Update ; updates pointers
    end
end
```

To prove correctness, consider points x, y, and z(see figure 5.6). Assume that the algorithm has worked up to the end of k^{th} iteration and that there is a point p which should cause z to eliminate itself when it checks y, but fails to do so. For this point p to cause z to be eliminated it would have to be in the region on or above the line l. This point could not be in region B, as z would have already checked that area. If this point had been above $l1$ then y would have

been eliminated and told z about the point. If y has not been removed, the only place for the point p must be in Region C, a contradiction to the existence of point p. □

The reduction technique in Theorem 4 can be used along with the above proof. This leads to:

Corollary 2 *Given a $N \times N$ 0/1 image, the extreme points of all the figures can be enumerated in $O(\log N)$ time using an $(N/\log^{1/2} N \times N/\log^{1/2} N)$-optical mesh.*

5.2.3 Distance Problems

Another interesting problem is to identify and to compute the distance to a nearest figure to each figure in a digitized image. In the following, we use the l_1 metric. However, it can be modified to operate for any l_k metric.

Theorem 6 *Given a $N \times N$ 0/1 image, the nearest figure to all figures can be computed in $O(\log N)$ time using an $(N/\log^{1/2} N \times N/\log^{1/2} N)$-optical mesh.*

Proof: This algorithm is made of two steps. In the first step, each black pixel finds its nearest neighboring black pixel which belongs to a different figure. In the second step, each figure finds its nearest neighboring figure by finding the minimum among all those values obtained for the PEs at the boundaries. This can be done in $O(\log N)$ time, using the techniques of [84], once the input has been reduced to match the number of processors. The following pre-processing algorithm is used to reduce the size of the problem to match the number of processors. The first step of this reduction procedure is to sequentially compute the nearest neighboring figures to all the figures in regions of $\log^{1/2} N \times \log^{1/2} N$.

The second step of reduction is executed using a uniprocessor within blocks of size $\log^{1/2} N \times \log^{1/2} N$. The following is an outline of this procedure. Complete code can be find in Appendix III.

```
procedure Distance.Main
    from l=1 to c do ; c = log^{1/2} N
        for direction=1 to 4 do; up, down, left, right
            call Distance ; distance is computed for each row/column
    continue l
end
```

The third reduction step is to merge the information at the block boundaries. This can be performed by using the merging procedure explained in Theorem 4. This is done to bring the block size to $\log N \times \log N$. □

5.3 Constant Time Geometric Algorithms

One for the most attractive properties of optics is superposition [50]. This property suggests that the resultant disturbance at any point in a medium is the algebraic sum of the separate constituent waves. Hence, it enables many optical signals to pass through the same point in space at the same time without causing mutual interference or crosstalk. Using this property, in [61] they showed how a single memory element can be read by many processors at the same time. In this section, we employ this characteristic to allow concurrent writes if all the requesting processors want to write a "1". This leads to constant running time of the following geometric algorithms, under the assumption that broadcasting can be done in unit time:

Lemma 4 *Given a* $(N^{1/2} \times N^{1/2})$ *image, using an* $(N \times N)$ *optical mesh, in* $O(1)$ *time,*

1. *For a single figure, its convex hull and a smallest enclosing box can be determined.*
2. *For each figure, the nearest neighboring figure can be identified.*

Table 5.1 shows a comparison of our results in computing geometric properties of images with those on mesh based architectures.

5.4 Parallel Implementation of Iterative methods for Sparse Systems

Solutions to many problems in image understanding can be posed in terms of iterative improvement to an initial configuration. For example, discrete relaxation based approaches to scene labelling can be viewed as an iterative improvement process. In such problems, the underlying graph is usually sparse. But this sparsity is not regular. Efficient parallel implementaions of such relaxation methods can be realized using the OMC with holographic connections. For sake of illustration, we consider the solution of a set of linear equations, characterized by sparsity, that is, the associated coefficient matrix contains a large proportion of zero elements.

There are two general classes of methods for solving sparse linear systems, direct and iterative. Direct methods usually involve matrix factorizations and lead to additional nonzero elements being created during the computation. These fill-in elements cause extra required storage and an increase in computation time. On the other hand, Iterative methods preserve the sparsity of the matrix during

computation and reduce the problem into some simple iterations of matrix-vector multiplications. They are preferred for solving large sparse systems because they can take advantage of zeros in the matrix and tend to be self-correcting and hence tend to minimize roundoff error.

When designing a special purpose sparse system solver, some important issues must be carefully considered. The architecture must be efficient for sparse matrix-vector multiplications and must realize any abitrary iterative matrix structure. Moreover, the structure of the iterative matrix is fixed throughout the computation. Thus, the expensive crossbar networks may be quite wasteful for this kind of computations. Even though electro-optical arrays have been designed for dense matrix computations and sparse banded matrix computations, no architectures are known for parallel solution to sparse linear systems in which the nonzeroes are arbitrarily distributed.

Codenotti and Romani [20] presented a modular VLSI structure which is capable of reconfiguring for different problem sizes. The main components in their design are an array of m PEs and a mesh of switch nodes where the switch nodes are configured according to the particular structure of the coefficient matrix. The time required for one iteration is $O(m)$ for solving a set of m equations.

In this section, we show how the proposed optical mesh can be used to provide an efficient iterative solution of general sparse linear systems. Any iterative matrix structure can be realized by the use of holograms. Although the reconfiguration time for the hologram can be in the order of seconds in the current technology, it only needs to be done once during the preprocessing phase in which the structure of the coefficient matrix is used to define the holographic connections.

The interconnection pattern remains the same throughout the computation. An optimal $O(\log m)$ time can be achieved by this design and the number of processors depends only on the number of non-zero elements in the matrix. This method is attractive when many computations are to be performed in which the structure of the coefficient matrix is fixed. It is well suited for implementation of many iterative methods such as Gauss-Jordan, Gauss-Siedel and the Conjugate method [30].

5.4.1 The Algorithm

Consider the iterative method

$$x^{k+1} = Mx^k + g$$

where M is sparse and nonsingular. Let n_i be the number of nonzero element in the ith row of M, and let $j_1, j_2, ..., j_{n_i}$ be the columns corresponding to these elements. Thus, the above equation can be rewritten as

$$x_i^{k+1} = \sum_{s=1}^{n_i} m_{ij_s} x_{j_s}^k + g_i.$$

Suppose there are n processors and each of the processors stores exactly one nonzero element in matrix M.

Theorem 7 *The OMC can be used to solve each iteration of the iterative solution to any general sparse linear systems with m variables and n nonzero elements in* $O(\log m)$ *time.*

Proof: Consider the following steps for solving the sparse matrix vector multiplication:

1. Send x_i to all processors with elements in ith column.

2. Perform multiplication in each processor.

3. Sum up all the products from the processors with elements in the same row.

The broadcast of x_i in step 1 is accomplished by sending the data to the first two processors with elements in ith column and then from those two processors to the next four processors with elements in the same column. It can be trivially shown that this step takes at most $O(\log m)$ time. The summation step can be similarly done by summing the products in groups of two processors with elements in the same row and then to the rest of the products. Therefore, the matrix-vector multiplication can be performed in $O(\log m)$. Since the iterative matrix is nonsingular, the x_i elements and g_i elements are stored in any one processor with a non-zero element in the same row. This processor is always the last one to be routed in the summation step. Thus we don't need extra processors to store x and g elements and extra connections to route the result back. After each iteration, a norm $|x_i^{k+1} - x_i^k|$, for all i, must be checked for convergence. If the maximal acceptable error is reached or the maximum allowable number iterations has been exceeded, the iteration process halts. The interconnection patterns can be reconfigured for different inputs and different sizes as long as the number of nonzero elements doesn't exceed the number of processors.

Problem	$(N \times N)$ Mesh of Trees [84]	$(N \times N)$ Mesh Connected Computer [76]	$(N - PE)$ Reduced Mesh of Trees [83]	$(N \times N)/\log N$ Optical Mesh
Labeling figures	$(\log^3 N)/\log\log N$	N	N	$\log N$
Convex hull of all figures	$\log^3 N$	N	N	$\log N$
Nearest figure to all figures	$\log^2 N$	N	N	$\log N$

Table 5.1: Time for computing some properties of images

Chapter 6

Non Von Neumann Computations

The sequential and parallel Von Neumann models have a number of common features: (1) data are passed indirectly between instructions via references to shared memory cells; (2) literals may be stored in instructions, which can be viewed as an optimization of using a reference to access the literal; (3) flow of control is implicitly sequential but explicit control operators can be used for parallelism, etc.; and (4) because the flows of data and control are separate, they can be made identical or distinct.

In this chapter, we concentrate on realizing models of computations that operate under principles that are unlike the ones specified above, using an optical model of computation. The control scheme in these models are distributed and is performed through interactive cooperation of the processing elements. Thus, an

efficient communication network is a crucial factor in their performance. The optical interconnection networks presented in the previous chapters can be naturally adopted for realization of these models.

6.1 Implementation of Neural Networks

Conventional computers, regardless of the Von Neumann shortcomings, are very successful in the solution of structured problems. Such problems are characterized by the fact that they are solvable by the repeated application of a sequence of operation, i.e., by an algorithm that may be defined in concise terms. However, not all problems are like that. There are a large number of problems that we do not know how to solve by a well defined, proven algorithm. At best we can suggest some sort of heuristic procedure. Examples for such problems include the recognition of some one's face, or the evaluation of the situation in a chess game.

If we try to analyze how we as humans solve these problems, it seems that we do so by drawing on a large bank of experience, accumulated by trial and error over the years. This suggests that computers with large memories and associative reasoning powers might be needed if we want to solve them mechanically. Models with such capabilities have been proposed recently, e.g. the Hopfield model of neural networks.

6.1.1 Interconnectivity in Neural Nets

Neural networks can be thought of as a special case of parallel computers; where neurons play the role of processors and synapses play the role of the communication links. The most striking difference between a neural network and an electronic circuit in a computer is degree of connectivity [88]. Optics is a natural context to contemplate the implementation of neural networks [5]. The highly communicative nature of the processing elements in neural models demand connection mechanisms that do not interact except at (or near) the processing elements. This is particularly true in systems which are globally connected. Light waves do not interact except under the attendance of a material medium, and then only weakly. In this section, we concentrate on the interconnection medium for neural networks, and propose an optical realization for such a model as shown in Figure 6.1. As shown, there are N receiving and N transmitting ports for each neural processor. Its operation is explained in the section below.

6.1.2 Operation

Compared to electronics, the computational limits of an optical neural network is superior but it's operation is identical [1]. The i^{th} neuron can be in one of two states: $u_i = -1$ (off) and $u_i = +1$ (on). The synaptic connections are undirected and have strengths which are fixed real numbers. In the optical version, these weights are carried with the optical beams. The neurons repeatedly examine their inputs and decide to turn on or off by the following scheme. Let w_{ij} be the strength (could be negative) of the synaptic connection from neuron j to neuron

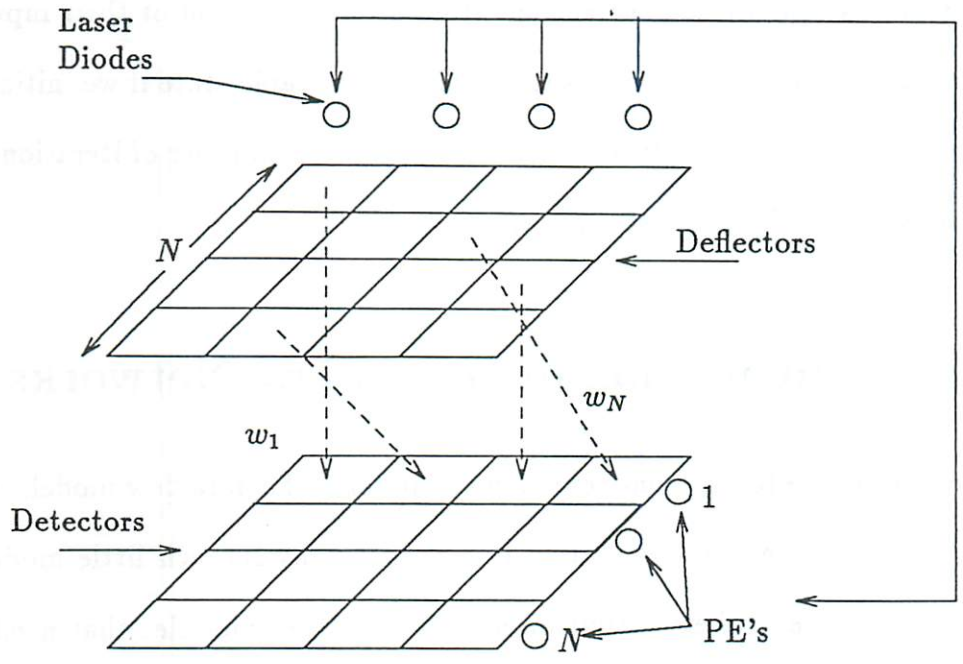


Figure 6.1: An implementation of a neural network

i. ($w_{ij} = w_{ji}$ and $w_{ii} = 0$). Let t_i be the threshold voltage of the i^{th} neuron. If the weighted sum over all of its inputs is greater than t_i, the i^{th} neuron turns on and its state becomes $+1$. If the sum is less than t_i, the neuron turns off and its state becomes -1. If the sum equals t_i, the neuron maintains its previous state. The action of each neuron simulates a general threshold function of $N-1$ variables (the states of the rest of neurons). This can be described by:

$$u_i = sign(\sum_{j=1}^{N} w_{ij}u_j - t_i)$$

where $sign(x)$ is $+1$ for positive x, and undefined for $x = 0$. The network starts in an initial state and runs with each neuron randomly and independently

revaluating itself. Often, the network enters a stable point in the state space in which all neurons remain in their current state after evaluation of their inputs. The basic operation of the network is to converge to a stable state if we initialize it with a nearby state vector. It has been shown that the number of iterations of a neural network can be at most a polynomial in N [1]

6.2 Implementation of Data Flow Networks

A possible alternative to the Von-Neumann computer is the data flow model. The idea here is simply that each operation is carried out by its own little module, and these modules send their output directly to the other modules that need to use this data. Thus, the data flows through the system in parallel, and it is used as soon as it is produced [22, 6, 40, 114]. Most of electronic computers follow the Von Neumann architecture and not the data flow architecture because data flow requires too much interconnections and operational over head.

In a data driven computation organization, the instructions passively wait for some combination of their arguments to become available. This implies a select phase, which (logically) allocates computing elements to all instructions, and an examine phase, which suspends nonexecutable instructions. In a conventional data flow multiprocessor, communication between two instructions is much quicker if these instructions are allocated to the same processing element. Thus a program may run much faster if its instructions are clustered to minimize communication traffic between clusters and each cluster is allocated to one processing element. While such an optimal allocation is a NP-complete problem, it

also does not lead to a pure data flow computation. The proposed unit-delay, free space, optical interconnection networks, can provide a pure implementation of data flow programs.

In this section, we briefly review how some existing data flow machines can be enhanced with the use of the optical model of computation. To illustrate our ideas, we consider Irvine data flow machine and the MIT data flow computer.

6.2.1 Irvine Data Flow Machine

The Irvine data-flow (Id) machine is motivated by the desire to exploit the potential of VLSI and to provide a high-level, highly concurrent program organization [6, 114]. The Id machine has a packet communication organization with token matching. It consists of N processing elements and an $N \times N$ communications network for routing a token from the processing element generating it to the one consuming it. For the enhanced Id machine, we propose an organization based on the optical model of computation. There are N processing elements which can intercommunicate in unit time using the free space optical beams. This is shown in Figure 6.2. The organization of each processing element basically remains as it was in the original Id machine, except the input and the output sections. These two sections have to be enhanced to be capable of transforming electrical signals to optical ones and vice versa.

Other unaffected sections are; the waiting-matching section, which forms data tokens into sets for a consumer instructions; the instruction fetch section, which fetches executable instructions from the local program memory; and the service section, which is the arithmetic logic unit.

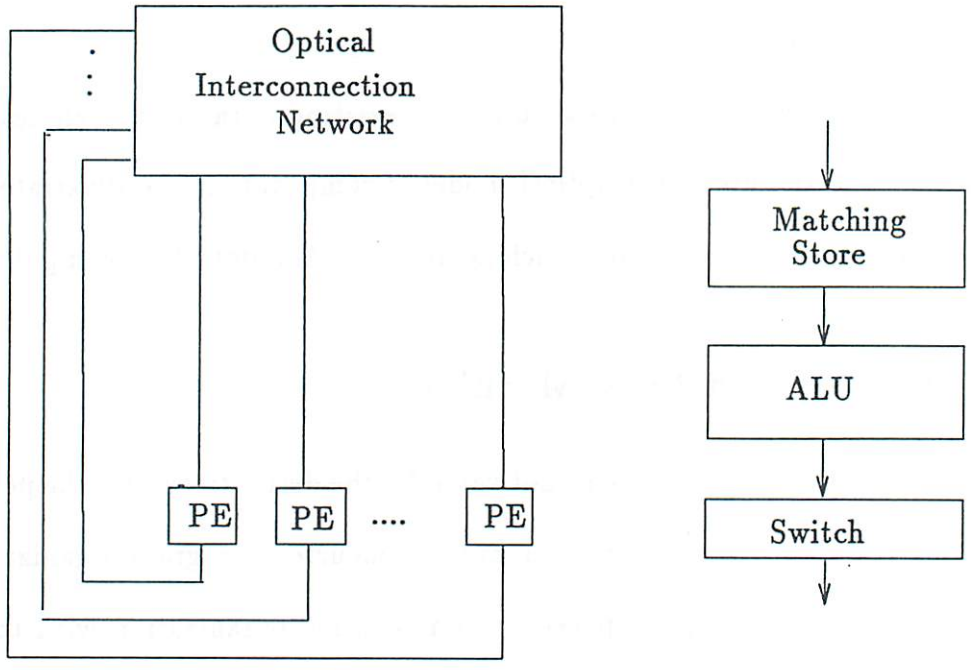


Figure 6.2: An enhanced Irvine data flow machine

6.2.2 MIT Data Flow Computer

The MIT data-flow computer [114, 22] consists of five major units connected by channels through which information packets are sent using an asynchronous transmission protocol. The five units are; the memory section, consisting of instruction cells that hold the instructions and their operands; the processing section, consisting of specialized processing elements that perform operations on data values; the arbitration network, delivering executable instruction packets from the memory section to the processing section; the control network, delivering control packets from the processing section to the memory section; and

the distribution network, delivering data packets from the processing section to the memory section. The enhanced MIT data-flow computer has all the above sections except that the arbitration network, control network, and the distribution network are all replaced by an optical interconnection network as shown in Figure 6.3. Other data-flow computers similarly can be enhanced using optical interconnection networks. For example, the enhanced Texas Instruments distributed data processor, can be reconfigured to have any topological organization in addition to be connected as a ring.

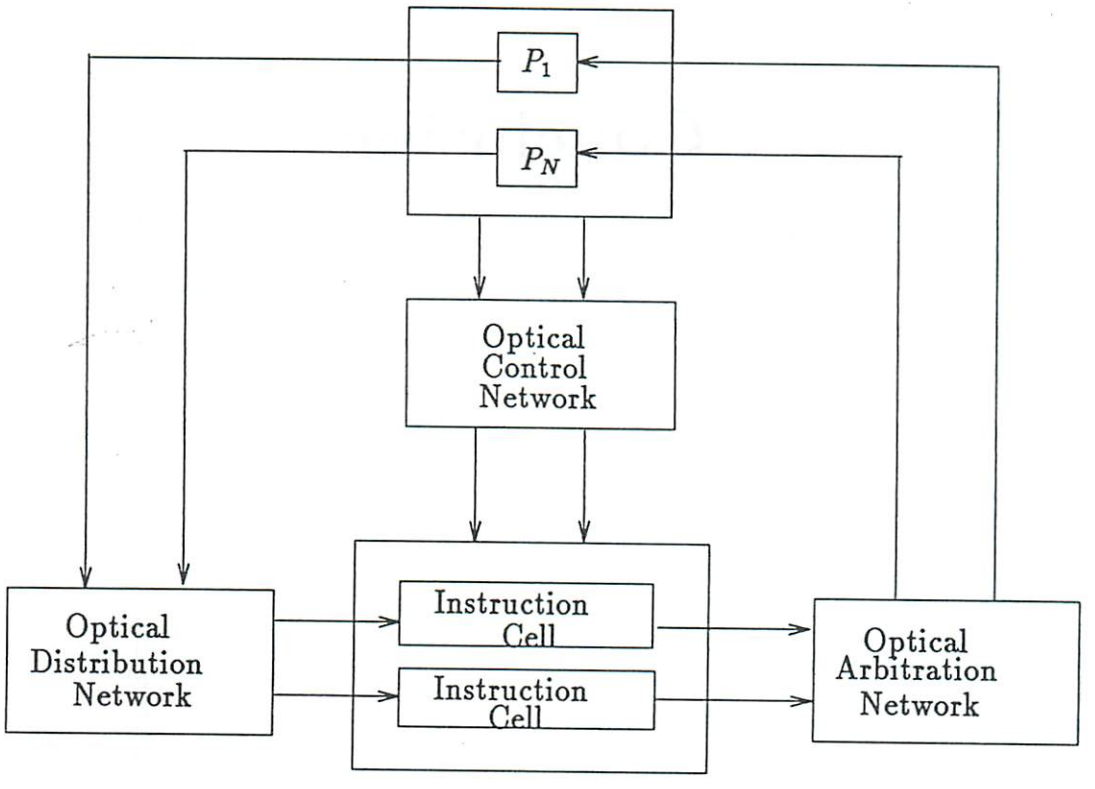


Figure 6.3: An enhanced MIT data flow machine

Part IV

Conclusion

Chapter 7

Conclusion

We began this thesis with a study of the requirements and capabilities of optics when used instead of electrical interconnects as a communication medium. We continued with a study of the computational limits in using optical interconnects under a proposed optical model of computation. This model uses the space above and around chips for interconnects, which places it at a comparable stage with the three dimensional VLSI models. In order to implement this model, we presented three possible physical architectures which achieve the unit time intercommunication delay assumed in PRAM. A direct simulation is possible using the proposed optical mesh using mirrors. A faster architecture is the optical array using acoustic optic devices with broadcasting capability. A considerably less expensive and currently implementable (using VLSI technology) design is the electro-optical crossbar. This can simulate fully connected networks with a reconfiguration time in the order of nanoseconds.

We studied the relationships between this model and PRAM, where unit cost

communication is possible, but there is a bottleneck in accessing memory modules. We presented a simple deterministic algorithm for efficient accessing of these memory modules. This algorithm simulates one step of an N processor EREW PRAM in $O(\log^2 N)$ time, and its probabilistic version has $O(\log N \log\log N)$ expected running time. The loosely synchronized pointer based techniques presented can act as general subroutines in simulating PRAM algorithms.

We showed the superiority of using optical interconnects by presenting efficient algorithms for applications such as finding the geometric properties of digitized pictures, digitized image template matching, and several problems in digital signal processing. Another applications in implementing non-Von Neumann computational models was also presented. We proposed an implementation of neural networks using a general purpose electro-optical crossbar which has the potential to interconnect each of neurons to all the others. Similar architecture can be modified to operate asynchronously for realization of a data flow model.

Appendix I

Variable Definition

x	a counter
b(x)	the value of element x
p	a pointer variable
q(x)	the two queues for the elements
l	a variable used in loop length
j(x)	the module number for element x
M(x)	the memory location for x
V(x)	the write value for x
z	another counter
p(x)	pointer to the next element

```
sort    for x = 1 to l
        v = b(x)
        if q2(v) = 0 then q2(v) = v
            else q1(p) = b(x); p = p+1
        next x

send    for x = 1 to l
        if q2(x) = 0 then goto loop
        M{x,j(x)} = V(x)
loop    next x

q2-q1   for x = 1 to l
        q1(x) = q2(x); q2(x) = 0
        next x

init    for x = 1 to log N
        p(x-1) = x
        next x

sort2   x = p(0)
        for z = 1 to l
        v = b(x)
        if q2(v) = 0 then q2(v) = v
            else q1(p) = b(x); p = p+1
        x = p(x)
        next z
```

```
send2       x = p(0)
            for z = 1 to l
            if q2(x) = 0 then goto loop
            M{x,j(x)} = V(x)
loop        x = p(x)
            next z

q2-q12      x = p(0)
            for z = 1 to l
            q1(x) = q2(x); q2(x) = 0
            x = p(x)
            next z

point       x = p(0); L = 0; r = 0
            for z = 1 to l
            if d(x) = 0 and r = 0 then goto loop
               else if d(x) = 0 then
                  p(L) = x
                  r = 0; L = x
               else r = 1
loop        x = p(x)
            next z
```

Appendix II

Variable Definition

x	x position of processor
y	y position of processor
p	pixel value(0/1)
e	square root of number of processors Note: assuming a square PE layout
k	number of the current processor(k=ey+x)
n	pointer next
b	boundary test; true=1
i	variable k
o	outer boundary test; true=1, false=-1
t	temporary n
l	counting variable
s	sum of turns this is used to determine inner/outer boundary statis
I	variable i
r	pointer right
p	pointer previous(Note this overwrites previous p)
Pm	maximum angle with pointer previous
Nm	maximum angle with pointer next
nm	current angle with pointer next
pm	current angle with pointer previous
D	eligibility text; true=0

```
procedure Multiply
        if x mod 2=0 then go to ymod
        if y mod 2=1 then p← p_{e+1}
           else p← p_1
        end
ymod if y mod 2=0 then end
           else p← p_e
end


procedure Border
        if k mod e=0 and k≠ e^2 then
```

```
down n← x+e(y+1)-k; end
          if k<e then
rite n← x+1+ey-k; end
          if k mod e=1 then
up n← x+e(y-1)-k; end
          if k>e²-e then
left n← x-1+ey-k; end
          case
              :p₁=0 and pₑ₊₁=1:go to rite
              :p=1 and pₑ=0:go to left
              :p=0 and p₁=1:go to up
              :pₑ=1 and pₑ₊₁=0:go to down
              :else:n← 0
          end
end

procedure Unify
          if n=0 then b=0; end
          i← k
          if n=e or n=1 then o← 1
              else o← -1
          t← n; l← 1; s← 0
          while l<log (e/2)² do
              if iₙ <i then i← iₙ
              if oₙ ≠ o then s← s+oₙ; o← s
              n← nₙ; l← l+1
          end
          n← t
          if s>0 then b← 1
              else b← 0
          if s<0 then b← -1
end

procedure Propagate
          I← i; l← 0
          while l<log (e/2) do
              if k mod 2n=0 then go to done
              r← k+1
              if iᵣ=i and i≠ 0 then go to done
              if i≠ 0 and b=1 and pᵣ=1 and bᵣ ≠ 1 then
                  iᵣ ← i; bᵣ ← 1
```

```
            r← r_r; l← l+1
        end
done if b=0 then i← I
end

procedure Check
        if b≠ 1 then end
        p_n ← k-n
        if (p+n) mod e=0 or p/e=n/e then D← 1
        l← 0; N← 0; P← 0
end


procedure Angle
        Pm← (P/e)/(P mod e)
        Nm← (N/e)/(N mod e)
        nm← (n/e)/(n mod e)
        pm← (p/e)/(p mod e)
        case
            :Pm>pm;pm>P:P← pm
            :Pm<pm;Pm>P:P← Pm
            :Pm>pm;Pm>P:P← Pm
            :Pm<pm;pm>P:P← pm
            :Pm=pm;pm>P:P← pm
            :Nm>nm;nm>N:N← nm
            :Nm<nm;Nm>N:N← Nm
            :Nm>nm;Nm>N:N← Nm
            :Nm<nm;nm>N:N← nm
            :Nm=nm;nm>N:N← nm
        end
        if N-P≤ 0 then D← 1
end


procedure Update
        N ← N_n; P ← P_p
        n ← n_n; p ← p_p
end
```

Appendix III

Variable	Definition
down	lb←1; le←c; ls←1; x←1; ko←c; k←l
left	x←4; ko←1; k←l×c
up	lb←c; le←1; ls←−1; x←3; ko←−c; k←c^2−c+l
right	x←2; ko←−1; k←l×c+c-1
x	x position within processor
y	y position within processor
N	number of processors
p	pixel value(0/1)
e	square root of number of processors Note: assuming a square PE layout
C	number of pixels in each processor
c	square root of C
k	number of the current position(k←cy+x)
n	pointer next
i	variable k; used as identifier
tn	temporary n
l,l2	counting variable
I	used as stack for i for non-boundary points
z,q	temporary variables
$d_{x,y,z}$	distance array; y←N x→ 1=up, 2=right, 3=down, 4=left z→ 1=distance, 2=position, 3=nearest 1 from the nearest 1 in opposite direction
ci	correct identification
b	boundary test variable
lb	loop beginning
le	loop ending
ls	loop skip variable
ko	k offset
d	distance variable
pd	previous distance
loop	main looping variable

```
procedure Border
        from l=1 to c^2 do
            k←l
            if p=0 then go to done
```

```
                if k=c^2 then n_-c ←c; n←-1; go to done
                if k mod c=0 and k≠c^2 then
                    if p_c=1 then
down                    n←c ; go to done
                    else goto left
                if k<c then
                    if p_1=1 then


right                   n←1 ; go to done
                    else goto down
                if k mod c=1 then
                    if p_-c=1 then
up                      n←-c ; go to done
                    else goto right
                if k>c^2-c+1 then
                    if p_-1=1 then
left                    n←-1 ; go to done
                    else goto up
                if p=1 and p_c=1 and p_c+1=1 and p_1=1 then goto done
                case
                    :p_c=1 and p_c+1=1    :n_c ←1
                    :p=1 and p_1=1        :n_1 ←-1
                    :p_c+1=1 and p_1=1    :n_c+1 ←-c
                    :p=1 and p_c=1        :n←c
                end
done        continue l
end



procedure Unify
            k←loop
            if n=0 then end
            if i≠0 then go to tn
            i←k
tn          tn←n
            do forever
                if i_tn <i then i←i_tn
                tn←n_tn+tn
                if tn=0 then exit
            end
```

```
            tn←n
            do forever
                i_tn ←i; tn←n_tn
                if tn=0 then exit
            end
end



procedure Edge
            z←loop; q←z mod c
            if q=0 or q=1 or z≤c or z>c²-c then r←true
            else r←false
end

procedure Propagate
            from l=1 to c²-c+1 by c do
                k←l
start           if k mod c=1 then go to done
                if n=0 then k←k+1; go to start
                ci←i
edge            k←k+1
                if k mod c=1 then go to done
                if i=ci then k←k+1; go to start
                i←ci; goto edge
done        continue l
end



procedure Distance
            d←0; if p=1 then p1←k
            from l2=lb to le by ls do
                if p=1 and i≠i_p1 then
                    d_x,k,1←d; pd←d; d←0; goto done
                if p=1 and i=i_p1 then
                    pd←pd+d; d_x,k,1←pd; d←0; goto done
                d_x,k,1←d
done            d←d+1; k←k+ko
            continue l2
end
```

Bibliography

[1] Y. S. Abu-Mostafa, "Neural Networks for Computing?", *Proc. of Conference on Neural Networks for Computing*, 1986.

[2] A. Agrawal and Lena Nekludova, "A Parallel O(log N) Algorithm for Finding Connected Components in Planar Images", *Proc. of IEEE International Conference on Parallel Processing*, 1987.

[3] A. V. Aho, J. E. Hopcroft, and J. D Ullman *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading Mass., 1974.

[4] R. Anderson and G. L. Miller, "Optimal Parallel Algorithms for List Ranking", Technical Report, Dept. of Computer Science, University of Southern California, 1987.

[5] J. A. Anderson, "Concepts in Connectionist Models", *Proc. of Conference on Neural Networks for Computing*, 1986.

[6] Arvind and K. P. Gostelow, "A computer capable of exchanging processors for time," *Proc. IFIP congress*, 849-854.

[7] R. Barakat and J. Reif, "Lower bounds on the computational efficiency of optical computing systems", *Journal of Applied Optics*, vol. 26, no. 6, March 15, 1987.

[8] T. E. Batchman, J. J. Sluss and D. Veasey, "An Introduction to Integrated Optics for Computing", *IEEE Computer*, December 1987.

[9] T. E. Bell, "Optical Computing: A Field in Flux", *IEEE Spectrum*, August, 1986.

[10] V. Benes, *Mathematical Theory of Connecting Networks and Telephone Traffic*, Academic Press, New York, 1965.

[11] G. Bergland, "Light Emitting Diodes", *Proc. of the IEEE*, Feb., 1972, pp.156-223.

[12] L. A. Bergman, W. H. Wu, A. R. Johnston, R. Nixon, C. C. Esener, S. C. Guest, P. Yu, T. J. Drabik, M. Feldman and S. H. Lee, "Holographic optical interconnects for VLSI", *Optical Engineering*, October 1986.

[13] L. Bic, "Data-driven logic: A basic model", *Proc. of the International Workshop on High Level Computer Architecture* 84, May 1984.

[14] M. A. Bobb, "Laser Diode Collimators for free Space Optical Communications", *SPIE Laser Diode Optics*, vol. 740, 1987.

[15] D. Casasent, "Guidelines for Efficient use of Optical Systolic Array Processors", *Proc. of SPIE International Optical Computing Conference*, 1983.

[16] D. Casasent, "Optical Artificial Intelligence Processors", *Proc. of SPIE International Optical Computing Conference*, vol. 700, 1986.

[17] H. Caulfield, R. Johnson and J. Shamir, "Massive Holographic Interconnection Networks and Their Limitations", To be Published in *Applied Optics.*

[18] A. Chu, D. Z. Tsang, D. L. Smythe and J. J. Lambert, "A Technology for optical Interconnections Based on Multichip Intergration", *Optical Engineering*, October 1986, vol. 25, no. 10.

[19] B. D. Clymer and J. W. Goodman, "Optical clock distribution to silicon chips", *Optical Engineering*, October 1986.

[20] B. Codenotti, F. Romani, "A compact and modular VLSI design for the solution of general sparse linear systems", *VLSI journal*, 5,1986, pp. 77-86.

[21] R. Cole and U. Vishkin, "Approximate Scheduling, Exact Scheduling and Applications to Parallel Algorithms", *Proc. of 27th IEEE Symposium on Foundations of Computer Science*, pp. 478-491, 1986.

[22] J. Dennis, "Data Flow Supercomputers", *IEEE Computer Magazine*, November 1980.

[23] C. R. Dyer, "A VLSI pyramid machine for hierarchical parallel image processing", *Proc. IEEE conference on Pattern Recognition and Image Processing*, 1981.

[24] M. Mary Eshaghian, D. K. Panda and V. K. Prasanna Kumar, "On the Hardware Requirement for 2-D Image Convolution in Electro-Optical Systems", USC-SIPI Report # 129, Dept. of Electrical Engineering, August 1988.

[25] M. Mary Eshaghian and V. K. Prasanna-Kumar, "VLSI Electro Optical Computers for Signal and Image Processing", *Proc. of Third International Conference on Supercomputing and Second World Supercomputer Exhibition*, 1988.

[26] M. Mary Eshaghian and V. K. Prasanna-Kumar, "Electro Optical Computers for Parallel Processing", *Proc. of IEEE International Symposium on Parallel Processing*, 1988.

[27] M. Mary Eshaghian and V. K. Prasanna-Kumar, "Parallel Geometric Algorithms for Digitized Pictures on Optical Mesh", *Proc. of IEEE International Conference on Frontiers of c Massively Parallel Computations.*

[28] M. Mary Eshaghian and V. K. Prasanna-Kumar, "An Implementation on Neural Networks using Optical Interconnects", *IEEE International Conference on Neural Networks*, 1988.

[29] M. Mary Eshaghian, Ho-In Jeon and V. K. Prasanna-Kumar, "Massively Parallel Architectures with Optical Interconnection Networks", *Proc. of ICO International Conference on Optical Computing*, Toulon, France, 1988.

[30] D.J. Evans, Ed. *Sparsity and its applications*, Cambridge University Press, Cambridge, London, 1985.

[31] G. Feinberg, "Light", *Scientific American*, pp 50-59, September 1968.

[32] D. G. Feitelson, "Optical Computers from a Computer Science Viewpoint", Technical Report 87-6, The Leibniz Center for Research in Computer Science, Institute of Mathematics and Computer Science, The Hebrew University of Jerusalem, May 1987.

[33] W. Feller, *An Introduction to Probability Theory and Its Applications*, John Wiley & Sons, 3rd ed., vol. 1.

[34] T. Y. Feng, "A Survey of Interconnection Networks", *IEEE Computer magazine*, December 1981.

[35] H. Freeman and R. Shapira, "Determining the Minimum-Area Encasing Rectangle for an Arbitrary Closed curve", *Communications of ACM*, vol. 18, no. 7, pp. 409-413, July 1975.

[36] J. A. Fried, "Optical I/O for high speed CMOS systems", *Optical Engineering*, October 1986.

[37] E. Friedrich, S. Valette and P.Gidon, "Optical Interconnections for WSI", *Proc. of ICO International Conference on Optical Computing*, Toulon, France, 1988.

[38] S. Fujita, R. Aibara, M. Yamashita and T. Ae, "A Template Matching Algorithm using Optically-Connected 3-D VLSI Architecture", *Proc. of 14th Annual International Symposium on Computer Architecture*, 1987.

[39] H. Gazit, G. L. Miller, and S. H. Teng, "Optimal Tree Contraction in the EREW model", Extended Abstract, Department of Computer Science, University of Southern California, 1986.

[40] J. L. Gaudiot and M. D. Ercegovac, "Performance evaluation of simulated data flow system using variable resolution actors," *Journal of Parallel and Distributed Computing*, November 1985.

[41] T. K. Gaylord and E. I. Verriest, "Matrix Triangularization Using Arrays of Integrated Optical Givens Rotation Devices", *IEEE Computer*, December 1987.

[42] P. Gidon, S. Valette and P. Mottier, "Integrated Lenses on Silicon Nitride Waveguides", *Optical Engineering*, vol.24, no.2, April 1985.

[43] J. W. Goodman, S. Y. Kung, F. J. Leonberger and R. A. Athale, "Optical Interconnections for VLSI systems", *Proceedings of IEEE*, vol.72, July 1984.

[44] H. P. Graf, L. D. Jackel, R. E. Howard, B. Straughn, J. S. Denker, W. Hubbard, D. M. Tennant and D. Schwartz, "VLSI Implementation of a Neural Network Memory with Several Hundreds of Neurons", *Proc. of Conference on Neural Networks for Computing*, 1986.

[45] R. I. Greenberg and Charles E. Leiserson, "Randomized Routing on Fat-Trees", *Proc. of the 26th Annual Symposium on Foundations of Computer Science*, pp.241-249, October 1985.

[46] D. G. Hall, "Survey of Silicon-Based Integrated Optics", *IEEE Computer*, December 1987.

[47] D. H. Hartman, "Digital High Speed Interconnects : A Study of the Optical Alternative", *Optical Engineering*, October 1986, vol. 25, no. 10.

[48] P. R. Haugen, S. Rychnovsky, A. Hussain, L. D. Hutcheson, "Optical Interconnects for High Speed Computing", *Optical Engineering*, October 1986, vol. 25, no 10.

[49] Kai Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing*, McGraw-Hill Inc., 1984.

[50] E. Hecht, *Optics*, 3rd ed., Addison-Wesley, 1987.

[51] R. Hecht-Nielsen, "Performance Limits of Optical, Electro-optical and Electronic Neuro-computers", Technical Report, TRW Rancho Carmel Artificial Intelligence Center, San Diego, CA. , March 13th 1987.

[52] R. Hecht-Nielsen, "Neuro-computing: Picking the human brain", *IEEE Spectrum*, March 1988.

[53] J. J. Hopfield and D. W. Tank, "Computing with Neural Circuits: A Model", *Science*, vol. 233, August 1986.

[54] A. Huang, "Architectural Considerations Involved in the Design of an Optical Digital Computer", *Proc. of IEEE*, vol. 72, no. 7, July 1984.

[55] A. Huang, "Parallel Algorithms Optical Digital Computers", *Proc. of International Optical Computing Conference*, 1983.

[56] H. Ito, N. Komagata, H. Yamada and Humio Inaba, "New Structure of Laser Diode and Light Emitting Diode Based on Coaxial Transverse Junction", Research Institute of Electrical Communication, Tohoku University, Sendai 980, Japan.

[57] B. K. Jenkins and C. L. Giles, "Parallel Processing Paradigms and Optical Computing", *Proc. Optical Computing Symposium*, SPIE vol. 625, January 1986.

[58] B. K. Jenkins, P. Chavel, R. Forcheimer, A. A. Sawchuck and T. C. Strand, "Architectural Implications of a Digital Optical Processor", *Applied Optics*, vol. 23, no. 19, October 1984.

[59] B. K. Jenkins, T. C. Strand, "Computer Generated Holograms for Space-Variant Interconnections in Optical Logic Systems", *Proc. of SPIE International Conference Computer Generated Holography*, pp. 110-117, August 1983.

[60] B. K. Jenkins, "Architectural Characteristics of Optical Logic Systems", *Proc. of the IEEE Computer*, 1985.

[61] B. K. Jenkins and C. L. Giles, "Superposition in Optical Computing", *Proc. of ICO International Conference on Optical Computing*, Toulon, France, 1988.

[62] B. K. Jenkins and C. L. Giles, "Complexity Implications of Optical Parallel Computing", *Twentieth Annual Asolomar Conference on Signals, Systems and Computes*, November 1986.

[63] F. Jenkins and H. White, *Fundamentals of Optics*, McGraw-Hill, 4th ed., 1976.

[64] H. I. Jeon and A. A. Sawchuk, "Optical Crossbar Interconnections using variable grating mode devices", *Applied Optics*, vol. 26, Page 261, Jan. 15, 1987.

[65] H. Jeon, "Digital Optical Processor Based on Symbolic Substitution Using Matched Filtering" Tech. Digest series, *OSA Topical Meeting on Optical Computing*, vol. 11, 1987.

[66] N. Kapany, "Fiber Optics", *Scientific American*, pp. 72-81, November 1960.

[67] M. K. Kilcoyne, S. Beccue, K. D. Pedrotti, R. Asatourian and R. Anderson, "Opto-electronic integrated circuits for high speed signal processing", *Optical Engineering*, October 1986.

[68] H. Kogelnik, "Limits in Integrated Optics", *IEEE trans. Microwave Theory and Techniques*, pp. 2-16, January 1975.

[69] N. Konforti and E. Marom, "Programmable Optical Interconnects", *Proc. SPIE 700*, pp. 209-213, July 1986.

[70] I. Lee, S. Goldwasser and D. Smitely, "Synthesis and mapping algorithms for a reconfigurable optical interconnection network", *Proc. of IEEE International Conference on Parallel Processing*, 1986.

[71] F. T. Leighton, "New lower bound techniques for VLSI", *IEEE FOCS*, 1981.

[72] A. W. Lohmann, "Chances for Optical Computing", *Proc. of International Optical Computing Conference*, 1983.

[73] G. L. Miller and J. H. Reif, "Parallel Tree Contraction and its Applications", *26th Symposium on Foundations of Computer Science*, pp. 478-489, 1985.

[74] R. Miller and Q. F. Stout, "Convexity Algorithms for Pyramid Computers", *Proc. of 7th International Conference on Pattern Recognition*, 1984.

[75] R. Miller and Q. F. Stout, "Efficient Parallel Convex Hull Algorithms", *IEEE transactions on Computers*, December 1988.

[76] R. Miller and Q. F. Stout, "Parallel Geometric Algorithms for Digitized Pictures on Mesh Connected Computers", *IEEE transactions on Pattern Analysis and Machine Intelligence*, March 1985.

[77] F. Morehead, "Ligh Emitting Semiconductors", *Scientific American*, pp. 108-122, May 1967.

[78] P. Mottier and S. Valette, "Integrated Fresnel Lens on Thermally Oxidized Silicon Substrate", *Applied Optics*, vol. 20, pp. 1630, May 1981.

[79] D. Nassimi and S. Sahni, "Parallel algorithms to set up the Benes permutation network", IEEE Transactions on Computers, vol. C-31, no. 2, February 1982.

[80] D. Nath, S. N. Maheshwari and P. C. P. Bhat, "Efficient VLSI networks for parallel processing based on orthogonal trees", *IEEE transactions on Computers*, 1983.

[81] M. H. Overmars and J. V. Leeuwen, "Notes On Maintenance of configurations in the plane", Technical report, Department of Computer Science, University of Utrecht, Netherlands, September 1980.

[82] V. K. Prasanna Kumar, "Communication Complexity of various VLSI Models", Ph. D. Thesis, Dept. of Computer Science, Pennsylvania State University, 1983.

[83] V. K. Prasanna Kumar and H. Alnuweiri, "Optimal Image Computations on VLSI Architectures with Reduced Hardware", *Proc. of IEEE Workshop on Computer Architecture for Pattern and Machine Intelligence*, 1987.

[84] V. K. Prasanna Kumar and M. Mary Eshaghian, "Parallel Geometric Algorithm for Digitized Pictures on Mesh of Trees", *Proc. of IEEE International Conference on Parallel Processing*, 1986.

[85] V. K. Prasanna Kumar and C. S. Raghavendra, "Array Processor with Multiple Broadcasting", *Proc. of the 1985 Annual Symposium on Computer Architecture*, June 1985.

[86] Russ Miller, V. K. Prasanna Kumar, D. Reisis and Q. F. Stout, "Data Movement Operations and Applications on Reconfigurable VLSI Arrays", *Proc. of IEEE International Conference on Parallel Processing*, August 1988.

[87] P. R. Prucnal, "VLSI Fiber Optic Local Area Network", *Proc. of International Optical Computing Conference*, 1986.

[88] D. Psaltis, "Optical Realizations of Neural Network Models", *Proc. of SPIE International Optical Computing Conference*, vol. 700IOCC-1986, 1986.

[89] D. Psaltis, C. Park and J. Hong, "Higher Order Associative Memories and their Optical Implementations", *Neural Networks*, vol. 1, 1988.

[90] D. Psaltis and N. Farhat, "Optical Information Processing based on an Associative-Memory Model of Neural Nets with Thresholding and feedback", *Optics Letters*, vol.10, pp. 98, January 1985.

[91] D. Psaltis and J. Hong, "Shift-invariant Optical Associative Memories" *Optical Engineering*, vol. 26, no.1, January 1987.

[92] M. Quinn, *Designing Efficient Algorithms for Parallel Computers*, McGraw-Hill Inc., 1987.

[93] A. G. Ranade, "How to Emulate Shared Memory", *Proc. of Annual Symposium on Foundations of Computer Science*, 1987.

[94] E. Rich, *Artificial Intelligence*, McGraw-HILL Series in A.I., 1983.

[95] A. L. Rosenberg, "Three-dimensional Integrated Circuits", *VLSI Systems and Computations*, Edited by H. Kung, R. Sproul and G. Steel, Computer Science Press, Rockville, MD, pp. 69-79, 1981.

[96] A. Rosenfeld, "Parallel Processors for Image Processing: 2-D arrays and extensions", *IEEE Computer*, January 1983.

[97] A. Rosenfeld, "The Prism Machine: An Alternative to the Pyramid", *Journal of Parallel and Distributed Computing 2*, pp. 404-411, 1985.

[98] A. Rosenfeld and A. Kak, *Digital Picture Processing*, (2 Volumes), Academic Press, 2nd ed., 1982.

[99] J. Rowell, "Photonic Materials", *Scientific American 255(4)*, pp. 124-134, October 1986.

[100] J. P. Sage, K. Thompson and R. S. Withers, "An Artificial Neural Network Integrated Circuit Based on MNOS/CCD Principles",*Proc. of Conference on Neural Networks for Computing*, 1986.

[101] A. A. Sawchuk, B. K. Jenkins, C. S. Raghavendra and A. Varma, "Optical Crossbar Networks", *IEEE Computer*, June 1987.

[102] A. A. Sawchuk, B. K. Jenkins, T. C. Strand, R. Forcheimer and B. H. Soffer, "Sequential Optical Logic Implementation", *Applied Optics*, vol. 23, pp. 3455, October 1984.

[103] A. Sawchuk, T. Strand "Digital Optical Computing", *Proceedings of the IEEE*, vol. 72, no. 7, July 1984.

[104] A. Schawlow, "Laser Light", *Scientific American*, Sept 1968, pp.120-136.

[105] L. J. Seigel, H. J. Siegel and A.E. Feather, "Parallel Processing approaches to image correlation", *IEEE Trans. on Computers*, vol C-31, March 1982.

[106] C. Seitz and J. Matisoo, "Engineering Limits on Computer Performance", *Physics Today*, pp. 38-45, May 1984.

[107] H. J. Siegel, *Interconnection Networks for Large-Scale Parallel Processing: Theory and case studies*, Lexington Books, Lexington, Mass, 1984.

[108] M. A. Sivilotti, M. R. Emerling and C. A. Mead, "VLSI Architectures for Implementation of Neural Networks", *Proc. of Conference on Neural Networks for Computing*, 1986.

[109] P. W. Smith, "Digital Optics: Progress towards practical applications", *Proc. of International Optical Computing Conference*, 1986.

[110] Y. Suematsue, "Advances in Semiconductor Lasers", *Physics Today #8(5)*, pp. 32-39, May 1985.

[111] S. L. Tanimoto, "A Pyramidal approach to Parallel Processing", *Proc. 1983 International Symposium on Computer Architecture.*

[112] C. D. Thompson, "Area time complexity for VLSI", *Proc. 11th Annual ACM Symposium on Theory of Computing*, pp. 81-88, 1979.

[113] C. D. Thompson, "A Complexity Theory for VLSI", Ph.D. Thesis, Dept. of Computer Science, Carnegie Mellon University, 1980.

[114] P. C. Treleaven, D. R. Brownbridge and R. P. Hopkins, "Data-driven and Demand-driven Computer Architecture", *Computing Surveys*, vol. 14, no. 1, March 1982.

[115] Jeffrey D. Ullman, *Computational Aspects of VLSI*, Computer Science Press, 1984.

[116] E. Upfal, "Efficient schemes for parallel communication", *JACM*, vol. 31, no. 3, pp. 507-517, July 1984.

[117] S. Valette, P. Mottier, J. Lizet, P. Gidon, J. P. Jadot and D. Villani, "Integrated Optics on Silicon Substrate : A Way to Achieve Complex Optical Circuits", *SPIE Integrated Optical Circuit Engineering III*, vol. 651, 1986.

[118] L. G. Valiant and G. J. Brebner, "Universal Schemes for Parallel Computations", *Proc. of ACM Symposium on Theory of Computing*, pp. 263-277, 1981.

[119] C. M. Verber, "Integrated-Optical Approaches to numerical Optical Processing", *Proceedings of the IEEE*, vol. 72, no 7, July 1984.

[120] M. Warren, S. Koch and H. Gibbs, "Optical Bistability, Logic Gating and Waveguide Operation in Semiconductor Etalons", *IEEE Computer*, December 1987.

[121] L. C. West, "Picosecond Integrated Optical Logic", *IEEE Computer*, December 1987.

[122] H. J. White and W. A. Wright, "Holographic Implementations of Hopfield Model with Discrete Weightings", *Applied Optics*, vol. 27, pp. 231, January 1988.

[123] B. Widrow and R. Winter, "Neural Nets for Adaptive Filtering and Adaptive Pattern Recognition", *IEEE Computer*, March 1988.

[124] J.C. Wyllie, "The complexity of parallel computations", Ph.D. dissertation, 1979, Dept. of Computer Science, Cornell University, Ithaca, Ny.

[125] A. C. Yao, "Some Complexity Questions Related to Distributed Computing", *Proc. 11th Annual ACM Symposium on Theory of Computing*, pp. 81-88, 1979.

[126] A. C. Yao, "The Entropic Limitations on VLSI Computations", *Proc. 13th Annual ACM Symposium on Theory of Computing*, pp. 308-311, 1981.

FAULT TOLERANCE AND RELIABILITY ANALYSIS OF LARGE-SCALE MULTICOMPUTER SYSTEMS

by

Walid A. Najjar

Technical Report No. CENG 89-01

A Dissertation Presented to the
FACULTY OF THE GRADUATE SCHOOL
UNIVERSITY OF SOUTHERN CALIFORNIA
In Partial Fulfillment of the
Requirements for the Degree
DOCTOR OF PHILOSOPHY
(Computer Engineering)

July 1988

To my parents

Acknowledgments

First and foremost, I would like to thank Professor Jean-Luc Gaudiot, my advisor, for the thorough guidance, constant support and encouragement he has provided throughout my years in graduate school. To him I express my deep feelings of respect and appreciation as a teacher, a researcher and a person.

I would also like to thank the members of my guidance and dissertation committees: Professors Kai Hwang, Dean Jacobs, Michel Dubois, Prasanna Kumar and Dr. David Mizell.

I would like to express my special gratitude to Dr. David Mizell for the constant support and valuable feedback he has provided.

My heartfelt thanks go to Dr. Fadi Kurdahi whose friendship and encouragements have been invaluable.

Finally, I would like to thank all the great friends and colleagues, both in the EE-Systems Department and at ISI, who have made these past years very enjoyable: Christoph Scheurich, Jean-Luc Jezzouin, Skevos Evripidou, Dr. Rivi Sherman, Steve Farnworth, Susan Coatney, Diane Demetras.

This work has been supported by the National Science Foundation under Grant No. CCR-8603772 and by the Office of Naval Research, Arlington, VA, under Contract No. N00014-86-K-0311.

Contents

List of Figures

List of Tables

Abstract

Recent developments in VLSI technology have made possible the design and implementation of massively parallel computing systems comprising several thousand processing elements. As the number of computing elements in a system increases, the likelihood of one or more elements failing, during a given time interval, as well as the complexity of the system level diagnosis increase. Fault-tolerance is, therefore, to become an integral part in the architectural design of large-scale systems and reliability an important measure in the evaluation of their performance.

This dissertation addresses the issue of the effects of increased processor failures rate in large-scale gracefully degradable distributed computing systems. A probabilistic model of network disconnection is developed and used to evaluate the effects of node failures on the network topology. The results show that although the probability of network disconnection decreases with increasing system size, the *resilience* of a given topology to network disconnection decreases when the connectivity is kept constant.

Combined measures of performance and reliability are used to evaluate the trade-off between increased computational power and failure rates as the number of processors is increased. It is demonstrated that, for a given recovery mechanism, there exists an optimal number of processors at which the amount of *reliable computational work* the system could deliver is maximum.

Finally, a simple distributed iterative algorithm for fault-tolerance is presented and evaluated. Based on a functional execution model of tasks, this algorithm allows the implementation of run-time fault detection, checkpointing and recovery.

Chapter 1

INTRODUCTION

> *The most constant difficulty in contriving the engine has arisen from the desire to reduce the time in which the calculations were executed to the shortest which is possible. It is not to be presumed that such an attempt has succeeded. How near the approach has been made must remain for aftertimes to determine.*
>
> Charles Babbage
> *On the Mathematical Powers of the Calculating Engine*

The rapid development of Very Large Scale Integration (VLSI) technology, in the last decade or so, has opened a wide spectrum of new possibilities in the design of computing systems. In fact, in the past ten years, we have witnessed the development of ever more powerful and sophisticated commercially available microprocessors. While the switching speed of these devices has improved by several orders of magnitude, it has becomes more difficult and costly to achieve orders of magnitude speed-ups by solely upgrading the logic technology. More emphasis is therefore being put on building parallel architectures to achieve faster program execution.

The availability, at low cost, of fast microprocessors and high density memory chips has opened the way for the design and development of Large-Scale or Massively Parallel computing systems where thousands of Processing Elements cooperate on the solution of a single problem. Such systems are becoming a cost effective alternative to conventional supercomputing for a wide variety of applications. In fact, their cost-performance is roughly estimated at $1,000 per MFLOPS whereas it is on the order of $10,000 per MFLOPS in the supercomputer family.

Multiprocessors are generally classified as being *tightly coupled* or *loosely coupled.* Processors in a tightly coupled multiprocessor system communicate through a globally shared memory. In loosely coupled multiprocessors, or multicomputers, processors have their own local memory and communicate by message passing over a shared interconnection network. The distinction between message passing and shared memory systems is not a very strict one. In fact, several schemes have been proposed that allow the implementation of a *logically shared* global memory on a physically distributed local memory system [Ran87]. The Denelcor HEP [Kow85] and the BBN Butterfly [BBN85a, BBN85b] implement such a model.

The objectives in using multiprocessor systems can either be a higher throughput or faster execution time of a single application program. Higher throughput can be obtained with general purpose time-shared multiuser multiprocessors such as the Sequent Balance or Symphony systems, the Encore Multimax, the Alliant FX8, the BBN Butterfly, etc. Multiprocessors such as the Intel iPSC, the NCUBE, the Connection Machine, etc. are mostly intended for faster execution of application programs.

Large-scale computing systems are targeted towards applications that exhibit a very large degree of parallelism. The execution of these applications often requires a very large amount (thousands) of CPU hours which can become prohibitively expensive even on high-speed uniprocessor systems. Such applications are typically found in physics (e.g. quantum chromo-dynamics), chemistry (e.g. molecular modeling), aeronautical engineering (e.g. fluid dynamics), civil engineering (e.g. seismic modeling), signal and image processing, computer simulation, artificial neural networks, etc. By applying thousands of Processing Elements to a single job, the execution time can be reduced to a manageable size thereby allowing a more effective use of computer models. It is expected that the availability of massively parallel computing systems would computer models to be applied to very large problem sizes and therefore open new possibilities and yet unforeseen potentials in many areas of scientific research and engineering.

1.1 Large-Scale Computing Systems

The recent developments in VLSI technology has accelerated research in the area of massively parallel or large-scale computing systems (LSCS). The salient features of such systems, as described in [KR86], are:

- A large number of independent Processing Elements communicating over a high-bandwidth interconnection network.
- Highly distributed control and operating system functions in the processing nodes.
- Highly parallel applications that are modeled as a collection of concurrent and communicating tasks.

A commonly accepted criterion for defining massively parallel architecture is that of 1000 or more Processing Elements irrespective of the size or complexity of the individual processor. The objective behind such a criterion is to quantify the degree of parallelism available in the machine.

Two computing systems have been developed that can be categorized, because of their size, as massively parallel systems. The Massively Parallel Processor (MPP) was developed by Goodyear Aerospace Corporation for the Goddard Space Flight Center under a contract from NASA for satellite image processing applications. Started in 1979, the MPP was historically the first machine in this category [Bat80]. The MPP is an SIMD two dimensional array processor consisting of 16K (128 x 128) single-bit processing elements. The system can be configured as a flat array, a cylinder or a torus. More recently, the Connection Machine (CM) was built by Thinking Machines Corporation under contract from DARPA. Originally intended as a large-scale connectionist MIMD architecture for the simulation of entity-relationship graphs [Hil86], the Connection Machine, in its present form, is an SIMD architecture. It consists of 16K to 64K single-bit Processing Elements organized in groups of 16. Each group of 16 PEs are implemented on a single chip together with the associated section of the interconnection network. Groups of 16 PEs are connected in a binary n-cube. The Connection Machine is the first commercially available Large-Scale Computing System (LSCS). However, both the Connection Machine and the MPP systems have an SIMD architecture that relies on a central controller, the host computer, for instruction scheduling and I/O operations. Therefore, they do not exactly fit the above definition of large-scale computing systems.

The major research issues that face the development of massively parallel systems can be summarized in three main categories:

1. *Programmability:* the programming and execution models necessary to build software tools for a very large number of concurrent tasks.

2. *Communicability:* the necessary underlying communication structures that allows for fast transfer of information from one point in the system to another.

3. *Fault-Tolerance:* the ability of the system to sustain and gracefully recover from faults and failures.

While the development and architectural design of parallel computing systems is quickly becoming an established field, the research and development of the software tools necessary to program these architectures still lags behind. All the commercially available multiprocessor and multicomputer systems are programmed using traditional sequential languages (mostly C and FORTRAN and, to a lesser extent, Lisp) that have been augmented with a set of library routines. These routines implement the necessary operations such as process spawning, synchronization or message-passing. It is, therefore, the programmer's task to explicitly state and structure the execution of the program in parallel. Notable exceptions in this respect are *Lisp and C* [1] that were developed for the Connection Machine. Although *Lisp and C* are designed as extensions to existing languages, they allow implicit data parallelism.

Since the programmer cannot explicitly specify the parallel execution on several thousands processors, the programmability of LSCSs becomes a crucial issue in their development. The major task, in this respect, is to provide programming and execution models that allow implicit parallelism to be detected at compile time and exploited at run-time. The research, in this area, has focused on four programming and execution paradigms:

- *Functional Languages:* based on Lambda calculus, these languages offer a highly formal approach to programming. Examples include Lisp and FL [Bac78]. Reduction architectures have been proposed for the direct execution of such languages [Mag80].

- *Object-Oriented Languages:* these are are less formally defined than functional languages. As common characteristic, they offer the possibility to define data structures and functions at a higher level of abstraction than conventional languages. Examples include: SIMULA, the Actors model as proposed by Agha [Agh85], C++ [Str86], Linda [ACG86].

- *Data-Flow Languages:* they are closely related to functional languages. The main difference being in their data-driven execution model. Examples include: SISAL [MSA*85], Val [AD79], Id [NPA86].

- *Logic Programming Languages* exemplified by Prolog, their paradigm is based on predicate calculus.

The large number of Processing Elements makes it impossible to organize a large-scale system around a physically shared global memory. Therefore, any realistic LSCS should be organized around an interconnection network that allows communication between processors or clusters of processors. While low connectivity graphs such as the array and the torus are suitable for SIMD computations where all communications are made to near neighbors, their large diameter would introduce unacceptable delays in MIMD computations. On the other hand, low diameter graphs such as the binary n-cube have a very high node connectivity. Therefore, in large systems, each node

[1] *Lisp and C* are trademarks of Thinking Machines Corporation.

would have a very large number of links which can result in a high implementation cost per node. A family of interconnection network topologies suitable for large-scale MIMD computations called hypernets have been proposed by Hwang and Ghosh [HG87]. These are non-regular hierarchically structured graphs that exhibit a low diameter while preserving a lower node-connectivity.

As the number of Processing Elements is increased, the likelihood of one or more elements failing, during any given time interval, increases accordingly. The failure of an element has repercussions not only on the program execution, but also on the communication structure. Due to the expected increase in failure rate as the system size is increased, it is imperative that LSCSs be designed to sustain and gracefully survive failures. Fault-tolerance considerations, therefore, do not affect only the design of the individual Processing Element, but also that of the underlying communication network. Furthermore, the reliability analysis of these systems becomes an important element in their overall performance evaluation. The fault-tolerance issue in LSCSs, and the related reliability and performance analysis are thus the topic of this dissertation.

1.2 Issues of Fault-Tolerance

The advent of LSCSs places the issues of fault-tolerant design and reliability evaluation of multiprocessor systems under a new light. In fact, the presence of a very large number of processing elements within one multicomputer system not only increases the computational power of the system but also the likelihood of one or more processors failing within a given time interval.

As the number of processors increases, the execution time of a given computation, and therefore the total system utilization time, decreases. Because of the added parallelization overhead, the increase in speed-up is at best a linear function of the number of processors. On the other hand, the overall failure rate increases with the number of processors and with the complexity of the underlying interconnection network with respect to the failure rate of a single processor using a comparable technology. This increase is at least proportional to the number of processors and results in a decrease of the expected time to first failure of the system. Since both the computing power and the failure rate of the system are determined by the number of processors, and since this number, in LSCSs, is expected to be very high, the trade-off between performance and reliability is very critical to the design and implementation of LSCSs. In fact, one can readily see that, for example, in a system with 4000 processors where the mean time to first failure of a single processor is on the order of 10^5 hours, the expected time to first failure of the system is on the order of 25 hours. This value might often be smaller than the expected execution time of some computations.

Therefore, if LSCSs are to become a viable alternative to supercomputers for the execution of massively parallel computations, then fault-tolerance must become a primary objective in their architectural design and reliability a key issue in their evaluation.

The primary objectives of a fault-tolerant design in LSCSs are therefore to provide:

- correct computational results in the presence of failures;
- system level diagnosis;

The secondary objectives are:

- to maximize computational throughput
- to minimize the hardware and software overhead necessary to implement reliable execution

However, LSCS have an inherent redundancy that can be exploited in order to allow the system to gracefully degrade in presence of faults by providing continued operation at reduced performance levels. How to exploit this redundancy in order to provide an optimal compromise between reliability and performance is still an open problem. This problem is actually two fold:

- at the hardware level, the issue is the detection of the failure of an element and its subsequent isolation from the rest of the system, thereby allowing continued operation.
- at the software level, the issue is to restore a previously saved globally consistent state of the program and resume execution on a reduced system with, possibly, a balanced reallocation of the computing load across the system.

Because of the distributed nature of LSCSs, the fault-tolerance and reliability of the interconnection network are determining factors in the overall reliability of the system. In fact, the fault-tolerant capabilities are based upon the exchange of information between processing nodes and assume the ability of the underlying communication structure to carry out the required transfer of information across the system. Several interconnection structures have been proposed and researched [WF84]. Network topologies can be classified in two categories:

- *static* topologies, where the switching nodes correspond to the processing nodes such as in the hypercube topology;
- *dynamic* topologies, where the processing nodes are located at the input and output nodes of the network such as in a cross-bar switch;

The routing control in the network can be either:

- *circuit switched* where a physical path is established between source and destination nodes;
- *packet switched* where data is sent in one or several packets that includes the destination address;

In the present state of technology, it seems that static, packet switched, networks offer the most suitable approach for large-scale multiprocessing. This does not exclude that future technology developments might allow mixed mode networks that could incorporate, at different levels, static and dynamic topologies as well as integrated data and circuit switching.

The required feature of an interconnection network from the standpoint of system fault-tolerance and reliability can be identified as [KR86]:

- *Robustness:* is the inherent redundancy of communication paths provided by the interconnection network.
- *Reconfigurability:* a related notion which measures the ability of the communication structure to adapt to link or node failures by providing reliable communication between processors.

Robustness and reconfigurability are the primary requirements on a fault-tolerant and reliable communication structure. A related requirement is high diagnosability level. Diagnosis refers to the process of determining the location of a fault in the system. Correct diagnosis of faults is necessary to allow proper reconfiguration and recovery actions to be carried out. In a distributed system, the system level diagnosis is implemented by a testing structure where each node is tested by all or a subset of its neighbors. The test results are then shared by subsets of nodes to reach a coherent

conclusion on the state of the system. Kuhl and Reddy [KR81] demonstrated several algorithms for performing distributed diagnosis that can be proven correct for a maximum number of faults. They also proved that the diagnosability level achievable by a distributed fault-diagnosis algorithm is a direct function of the interconnection topology and the testing structure independently of any algorithm.

1.3 Dissertation Goal and Outline

The main goal of this dissertation is to study the issues of fault-tolerant designs and reliability analysis in large-scale multicomputer systems. The main characteristic that differentiates large-scale systems (with thousands of Processing Elements) form existing multicomputers or multiprocessors (with less than a hundred Processing Elements) is the system size. The impact of a very large system size on different measures of fault-tolerance will be the main focus of analysis throughout this dissertation.

The rest of this dissertation is organized as follows.

Chapter 2 presents a statement of the problem addressed in this dissertation as well as a review of the traditional approaches to fault-tolerant design and reliability evaluation. Proposed combined measures of performance and reliability are reviewed. The concept of distributed fault-tolerance is introduced. Finally, a summary of the major contributions made in this dissertation is presented.

Chapter 3 proposes a probabilistic approach to the evaluation of network fault-tolerance. A stochastic simulation of network disconnection is used to derive an approximate analytical model. The analytical model is then justified and verified for a family of regular graph topologies. *Network Resilience* is introduced as a new measure of the robustness attribute of interconnection networks and used to compare a number of regular and non-regular graph topologies.

Chapter 4 presents a computation oriented reliability evaluation of large-scale gracefully degradable systems. The measure of *Reliable Computational Work* is defined and used to demonstrate the non-scalability of graceful degradation.

Chapter 5 describes a methodology of functional program execution that allows asynchronous run-time checkpointing. An iterative distributed algorithm for fault-tolerance is proposed that allows for protection against failures as well as fault-detection. The performance and reliability improvements brought by this algorithm are evaluated and compared to other algorithms.

Concluding remarks and directions for future research in this area are presented in Chapter 6.

Chapter 2

PREVIOUS RELATED WORK

> *Two are better than one; because they have a good reward for their labor. For if they fall, the one will lift up his fellow: but woe to him that is alone when he falls; for he has not another to help him up.*
>
> Ecclesiastes, 4, 9-10

The design of fault-tolerant systems can be traced to the early years of computing systems design when John von Neumann, in 1956, described how redundancy can allow reliable systems to be built from unreliable components [vN56]. Since then, the evolution of fault-tolerant and reliable systems design has paralleled that of computer architecture through several generations of design technologies.

The research in this field has led to a better understanding of the types and characteristics or faults and failures that can affect the operation of a computing system [SS82]. Three major categories can be outlined according to the origin of the fault or failure, as:

1. *Design errors:* these are either hardware or software design errors that are more manifest in the prototyping and initial stages of a product. However, some of these might not be detected before years of utilization.

2. *Component failures:* an electronic component might fail because of overheating, current over-driving, aging, etc. They generally result in a permanent failure of the system.

3. *Transient faults:* these are intermittent faults of random nature whose effect, as opposed to the previous two types, are not reproducible. They originate from non-controllable sources such as cosmic radiations, electromagnetic noise (particularly in pulse form), etc.

Studies have shown that non-permanent faults account for 90 to 98 % of all detected faults [MST79]. Furthermore, Fuchs *et al.* argue in [FAH83] that the increase in integration level, results in reduced voltage levels and a subsequent reduction of noise margins which might increase the susceptibility of VLSI circuits to electromagnetic noise interference.

The purpose of a fault-tolerant design is therefore twofold:

1. To guarantee continued system operation in the presence of failures. This implies the detection of, and subsequent recovery from, permanent failures.

2. To provide a high probability of correct computational results and therefore the protection against non-permanent faults.

Redundancy, in various forms, has been the basis of most, if not all, fault-tolerant designs techniques and methodologies. In general, redundancy techniques call for the multiple execution of a computation; the majority result is then taken to be the correct result. Redundancy techniques can be applied either in *time* or *space*. In time redundancy, the same computation is executed repetitively until a majority result is obtained [PF82]. While this technique eliminates the effects of intermittent or transient faults, permanent faults cannot be detected unless different hardware units are used for the different executions. Space redundancy, commonly known as N-modular redundancy (NMR scheme), calls for N copies of a computation to be executed, concurrently, on N distinct hardware elements. The majority result is determined using a voting scheme. In order to guarantee a strict majority, N is usually chosen as an odd integer. Several variants of the N-modular redundancy scheme, such as reconfigurable NMR, hybrid redundancy and backup sparing, are described in [BF76, SS82].

In this chapter we review the main concepts and ideas in fault-tolerant design and reliability evaluation that relate directly to the design and evaluation of LSCSs.

2.1 Traditional Approaches to Fault-Tolerance

With the evolution of computing systems, fault-tolerant design has developed along two distinct directions satisfying the following objectives:

1. *Mission-Oriented Applications* where the objective is to guarantee a system reliability above a given minimum level for the duration of a mission. Such systems are often called *ultra-reliable* and are intended for situations where repair is not possible such as in space and military applications.

2. *Availability-Oriented Applications* where the objective is to maximize the percentage of time the system is up. These systems, therefore, aim at minimizing the frequency and duration of necessary repairs. The main applications of such systems are in On-Line Transaction Processing (OLPT) such as used in banking transactions and airline reservations. Historically, the earliest systems designed for high availability are computer-controlled electronic switching systems used in telephone networks.

Examples in the first category are the STAR [A*71] developed at the Jet Propulsion Laboratory for NASA. The STAR is a self-testing computer system that implements repair by switching in redundant spare units. The FTMP [HSL78] and the SIFT [WLG*78] were two concurrent projects aimed at designing an ultrareliable airline guidance system with a failure rate lower than 10^{-9}. The FTMP employs a form of redundancy related to the TMR-Hybrid redundancy called Parallel-Hybrid redundancy, in which each major module can substitute for any other module of the same type. The SIFT is also based on the replication of basic components but relies upon the software to detect and analyze errors and to dynamically reconfigure the system to bypass faulty units.

Several high-availability systems that fall in the second category are now commercially available [Ser84]. Examples include: the Nonstop system from Tandem Computers, the Synapse N+1 from Synapse Computer, etc. The ESS, from AT&T, is intended for the control of electronic switching systems [Tro78]. Based on a dual-processor architecture, the ESS is designed for a requirement of less that two hours of downtime in a 40 years period.

In both of these approaches redundancy is extensively used to improve the reliability, and/or availability, of the system. In LSCSs, the available hardware redundancy is aimed primarily at

increasing the performance of the system. However, because of the potential for an increased rate of failures in the system, this redundancy can be used to provide continued system operation in the presence of failures, albeit, at the cost of a decrease in system performance, thereby allowing the system to gracefully degrade.

2.2 Network Fault-Tolerance

Because of the highly distributed nature of LSCSs, the reliability of the communication structure acquires an increased importance. In fact, the loss of communication between any two nodes might halt the execution as well as impair the system's ability to rollback and recover. The design and performance analysis of interconnection networks for multicomputer systems has been a very popular area of research. Extensive research has been done in the design of high performance connection networks for very large number of processing nodes [Fen81]. The emphasis, in these designs, is on high performance, reliability and diagnosability of the systems as allowed by the interconnection network.

As outlined in section 1.2, the basic requirements on an interconnection network, from the system fault-tolerance standpoint are: (1) robustness and (2) reconfigurability. Robustness is a notion tightly related to the inherent redundancy available in the network topology itself. For example, a simple ring topology would allow up to two, non-adjacent, node or link failures after which the system is partitioned and routing across the system is impaired. Reconfigurability, on the other hand, relates to the ability of the routing algorithm to exploit the inherent path redundancy and provide reliable communication between processor pairs in the presence of node or link failures.

Several protection schemes specific to some network topologies have been proposed and analyzed. Raghavendra *et al.* [RAE84] propose a reconfiguration scheme for binary-trees based on spare Processing Elements, the number of spares being $\log N$ where N is the number of active Processing Elements. Rennels [Ren86] proposes a similar dynamic strategy for binary n-cube topologies (hypercubes). Fortes and Raghavendra [FR85] describe a dynamic reconfiguration scheme, with no spares, for array architectures, based on alternating row and column elimination. Pradhan, [Pra85b, Pra85a], describes a class of reconfigurable and optimally fault-tolerant network architectures. Similar networks have been described in [EH85] and [SSB87].

When the application is not topology specific, the loss of a Processing Element can modify the network configuration. Since this configuration is not, in the general case, a fully connected graph, successive failures can eventually result in a disconnection of the system, and therefore prevent some processors from communicating to some other processors. In systems that rely on a distributed fault detection, recovery and restart procedure, the connectivity of the graph is crucial to the success of this procedure. Pradhan has proposed the measure of *network fault-tolerance* as the number of processors that can fail while preserving graph connectivity [Pra85b]. Therefore, in a regular graph with connectivity n, the network fault-tolerance is $(n-1)$. In the general case of a non-regular graph where the connectivity of node i is denoted by n_i, network fault-tolerance can be defined as:

$$NFT = \min_i(n_i)$$

Network fault-tolerance is a very conservative measure of robustness that does not take into account the size of the system and therefore the actual *probability* of occurrence of a network partition as a result of node or link failures.

2.3 Evaluation Measures

The traditional fault-tolerance measures of reliability, availability and mission-time do not provide an adequate evaluation of degradable systems. These have been devised for ultrareliable (mission-oriented) or highly available systems and therefore fail to take into account the capability of multiprocessors to provide continued operations at lowered performance levels.

New measures, therefore, have been devised that provide a combined performance/reliability evaluation of degradable fault-tolerant computing systems. A conceptual framework of composite performance and reliability measures was first proposed by Meyer in [Mey80]. This framework is general enough to accommodate all possible forms of *performance accomplishments* or *rewards* such as throughput or computational work.

The performability of a system S is formally defined with respect to a set of accomplishment levels A as follows [Mey80]:

Definition 2.1 *If Y_S is a performance measure of a system S taking values in the set A, then the performability of S with respect to a subset B of A ($B \subseteq A$) is defined by the function $p_S(B)$ as:*

$$p_S(B) = Prob(\{\omega \mid Y_S(\omega) \in B\})$$

In other words, given a performance measure (such as throughput) denoted by Y_S with a range of possible values A. The performability of the system S is the probability that the performance measure of the system after an event ω will be in a subset of A denoted by B. As an example, the performability of a degraded system could be the probability that its throughput be larger than 75% that of the undegraded system.

Closed-form solutions methods of performability for different types of systems have been proposed in [Mey82, FM84, DB87]. More recently, Smith *et. al.* have proposed a general framework of performability models based on Markov Reward Models (MRM's) [STR88].

Beaudry proposed the measures of *Computational Availability*, which is the expected value of the computation capacity of the system and *Capacity Threshold* which is the time when a specific value of the computational availability is reached [Bea78]. Performability and computational availability are measures that evaluate the computational capacity of a system in time and therefore are ideal for throughput oriented applications such as on-line transaction processing.

Also proposed by Beaudry is the *Computational Reliability* measure, which is the probability that at a given time the system correctly executes a given task, and the *Mean Computation Before Failure* ($MCBF$) which is the expected amount of computational work the system can deliver before the first failure, [Bea78]. These two measures are more suitable for *computation-oriented* applications where the critical issue is the *reliable completion* of the computation as is the case with massively parallel applications.

2.4 Distributed Fault-Tolerance

Since the very large number of PEs in a LSCS precludes the reliance on any physically shared global resource such as memory or controller, it implies that no central device can be used for the testing and monitoring of the system. Therefore, due to the distributed nature of LSCS, the task of identifying and isolating faulty PEs must itself be distributed throughout the system. In fact, a central controller or monitor would not only be a single point of failure in the system but might also become a performance bottleneck due to the large number of PEs. Preparata *et*

al. [PMC67] proposed a fault-tolerance algorithm where diagnosis methods are used to detect faulty elements that are subsequently isolated from the system. Based on this scheme, Kuhl and Reddy introduced the notion of *distributed fault-tolerance* as an attempt to allow the diagnosis and subsequent isolation of failed processors without relying on any central controller or memory system [KR80]. A distributed fault-tolerance allows for the graceful degradation of a system by providing for:

1. the detection of failures through system level diagnosis,
2. the isolation of the failed element from the rest of the system through logical and/or physical system reconfiguration,
3. the recovery from the failure and the restart of the computation. This step assumes the existence of some form of checkpointing that allows the rollback to a safe state in the computation.

For a survey of fault-tolerance issues in large-scale systems, the reader is referred to [KR86].

Several fault-tolerant schemes have been proposed that provide correct operations and allow the detection of faulty elements. In the general case of unstructured systems, the problem of *distributed* system diagnosis is known to be NP-complete. The system is therefore partitioned into a set of more manageable subsystems [Mal80, MM82]. A system is said to be t-diagnosable when up to t faulty elements can be detected, a comparison of modularly redundant and t-diagnosable systems is given in [CH81]. The proposed scheme is based on the assumption that each task is equivalent to a complete test (hundred percent fault coverage) of a processor. A recursive fault-tolerant algorithm, based on combined space and time redundancy, is proposed in [Agr85]. In this algorithm a task is initially executed on two processors and their results are compared, in the case of a mismatch, the task is executed on a third processor and the results are compared to the previous two, this process is recursively repeated until a match occurrs. The redundancy in this case is used to obtain a *plurality* vote. This algorithm allows for fault detection and correct task execution while maintaining a high system throughput. This scheme, however, relies on central commonly shared units such as a scheduler and signature processor as well as on recursively built common data structure that holds all the previous results pertaining to each task.

The Token Resending scheme, proposed by Gaudiot *et al.* [GR85], is based on the full replication of data and task restarting in a data-driven execution model. This scheme exploits the functional properties of data-flow execution to insure the retriability of tasks. Input data tokens to any actor are preserved in duplicates until the acknowledgement of the successful completion of that actor is received. This scheme, therefore, provides a the possibility of run-time checkpointing. Its implementation on a Tagged-Token data-flow architecture is described in [NG87a]. These features will be discussed in more details and exploited to implement distributed fault-tolerance in Chapter 5.

A similar scheme for distributed recovery has been proposed by Lin and Keller [LK86] for applicative program execution. It assumes a tree-structured execution of program tasks where no cycles are allowed.

2.5 Contributions of this Dissertation

In addressing the general issue of fault-tolerance and reliability analysis in large-scale multicomputer systems, this dissertation makes the following contributions:

1. *Disconnection probability.*
   An analytical model is devised that evaluates the probability of a network partition occuring as a result of multiple node failures. This model is verified by simulation results.

2. *Measures of network fault-tolerance.*
   Two new measures of network fault-tolerance are introduced: *Network Resilience* and *Relative Network Resilience*. These measures allow a comparison of different network topologies based on the cumulative probability of network disconnection.

3. *Effects of system size.*
   An analysis of the effects of a very large system size on reliability measures is presented. We demonstrate, analytically, the non-scalability of graceful degradation in large systems.

4. *Computational mesures of reliability.*
   The measure of computational work is used to evaluate the performance of a system as function of its size. The results show that an increase in system size might result in a decrease in the probability of safe completion of a lengthy computation.

5. *Distributed chekpointing.*
   A scheme, based on functional task execution, is proposed that allows the run-time checkpointing of programs with minimal overhead.

6. *Distributed fault-tolerance algorithm.*
   Based on the checkpointing scheme, an algorithm is proposed that implements distributed fault-tolerance as well as system level diagnosis. Its performance evaluation shows it to be superior to similar algorithms under normal conditions.

Chapter 3

A PROBABILISTIC EVALUATION OF NETWORK FAULT-TOLERANCE

"Look, Dave ... If you check my record, you'll find it completely free from error."

"I know all about your service record, Hal-but that doesn't prove you're right this time. Anyone can make mistakes."

"I don't want to insist on it, Dave, but I am incapable of making an error."

Arthur C. Clarke
2001: A Space Odyssey

The interconnection network is a central and major feature of any large-scale system design. Its characteristics determine the speed and volume of possible data communication which, in turn, determines its expected performance. They also determine the size of a realizable system in a given technology. As noted in Chapter 1, a large system size increases the probability of failures. Multiple node or links failures in a communication structure that is not based on a fully connected graph (as in a single bus based system) might result in a partitioning of the system. This event not only prevents the transfer of information between any two pairs of processors, but would also impair the process of distributed recovery and therefore result in a total system failure. In this chapter we propose a probabilistic approach to the analysis of network disconnections in distributed systems. The approach is based on a mixed analytical and simulation evaluation of the disconnection probability in a given network topology.

The assumed system and processor model is shown in Figure 3.1. It consists of a set of processing elements (PEs) communicating over an interconnection network. Each PE has a finite set of individual links to the network. The major components of a PE are a local memory system, a Processing Unit (PU) and a Communication Unit (CU). All communications from the PE to the outside world are handled by the CU over the set of links. In the analysis that follows, we will assume that the failure of either the PU or the CU is equivalent to the failure of all the links and therefore results in a complete PE failure. Furthermore, we will assume that the failures are uniformly and independently distributed throughout the system. This means that all processors are equally likely to fail and that the location of failures are uncorrelated.

In this chapter we will evaluate the disconnection probability in a family of regular graphs. The measure of *Network Resilience* is introduced as a probabilistic alternative to the static measure of

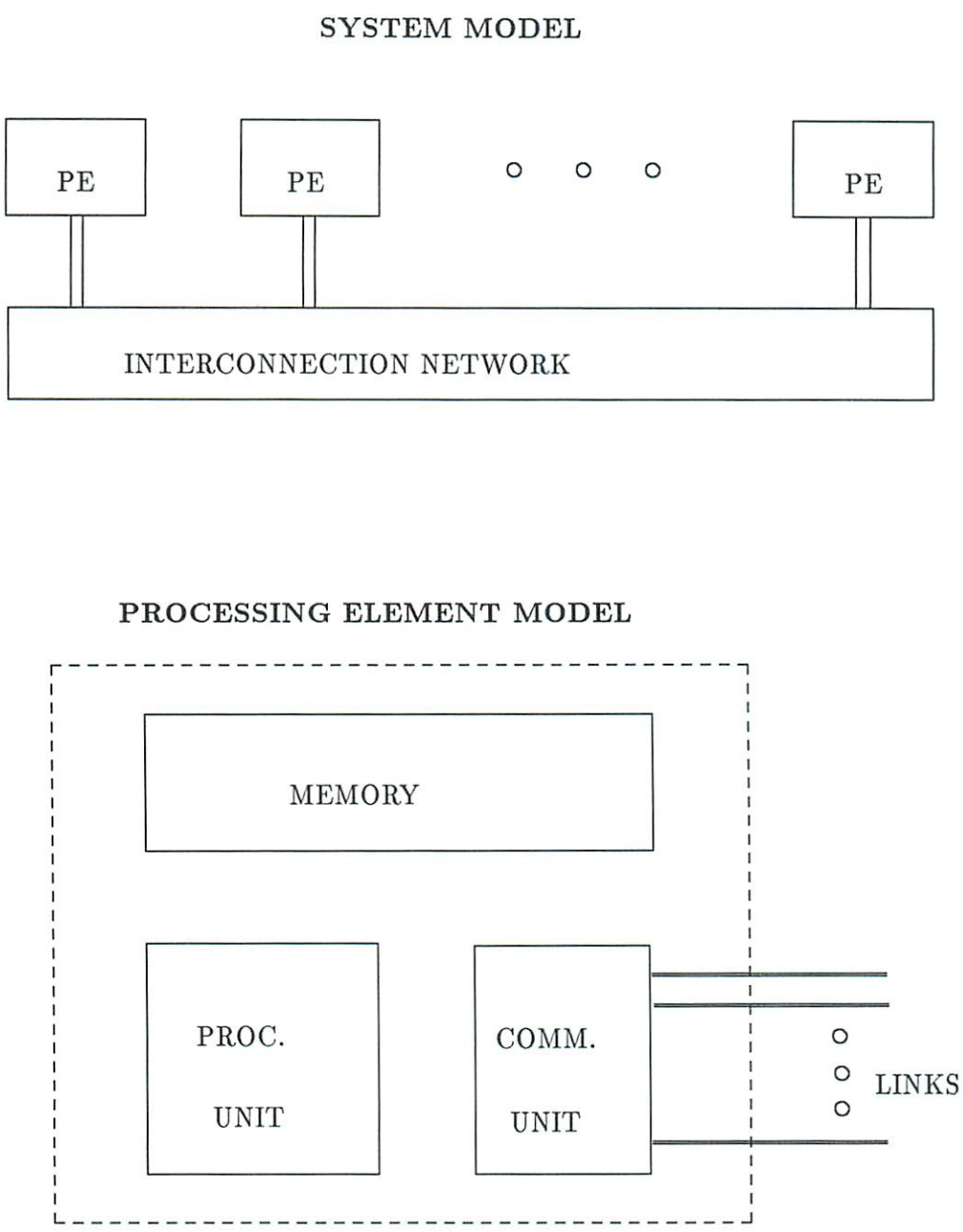


Figure 3.1: System and Processor Models

Network Fault-Tolerance. The obtained results are shown to be extensible, under certain conditions, to non-regular graphs. Finally, we demonstrate the analogy between node and link failures with respect to graph partitioning and extend the obtained results to include link failures.

3.1 Disconnection Probability

In this section we propose a combination of analytical and experimental approaches to the evaluation of the disconnection probability in a given network topology. Our model is a homogeneous, non-reconfigurable multiple processor system based on a class of n-regular graph network topology. An n-regular graph is a graph where the degree of all nodes is constant and equal to n. We assume that the loss or failure of a node implies the loss of all its connecting links.

The topological effects of a link or node failure are shown in Figure 3.2. Note that from the standpoint of node A, the loss of node B (Figure 3.2 b) or the link [AB] (Figure 3.2 c) have the same effects and the same potential of disconnecting node A from the rest of the system. However, the loss of node B has the same effect on three other nodes besides node A, while the loss of the [AB] link affects nodes A and B only.

After introducing the notations and formal definitions we analyze the initial disconnection probability and show that the case case of the single node disconnection is the most probable. These results are confirmed experimentally using a Monte-Carlo simulation. Based on these results, we propose an analytical approximation to the disconnection probability. Analytical and simulation results are compared and contrasted.

3.1.1 Notations and Definitions

Let $G(N, E)$ be an n-regular graph, with N nodes and E edges. A K-cluster is any connected subset of K nodes in $G(N, E)$. V_K is the number of neighbor nodes to a K-cluster and S_K is the number of K-clusters in $G(N, E)$.

Definition 3.1 *A system is in a disconnected state if and only if there exists a cluster of size K that is disconnected from the system and $K \geq 1$.*

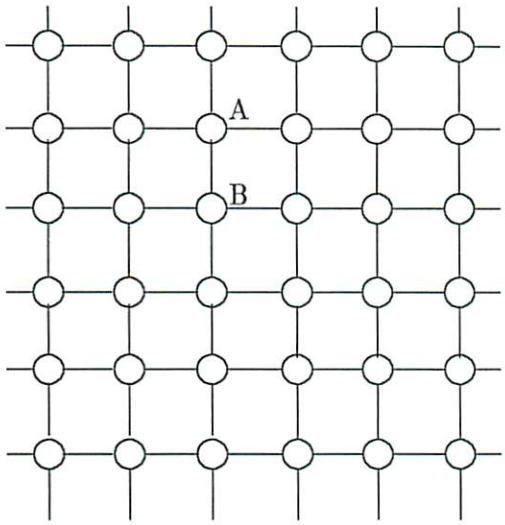
The analytical evaluation of the disconnection probability assumes a graph topology that satisfies the following two properties:

Property 3.1 *Let κ be the size of the mincut set of an n-regular graph $G(V, E)$, then $\kappa = n$.*

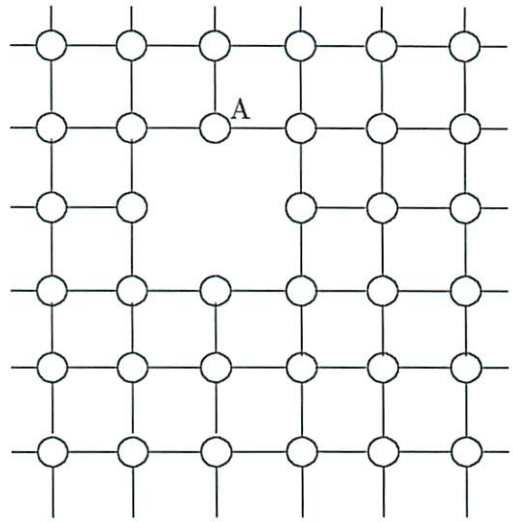
Property 3.2 *Given a K-cluster in $G(V, E)$, $1 < K < N/2 \Rightarrow V_K > n$.*

These definitions state that the size of the mincut set is equal to the node degree n, and the number of neighbors of a K-cluster is larger than n for $K > 1$. Therefore, the ring which is a regular graph, does not satisfy the stated properties. In this study, we consider, as examples, three popular topologies that satisfy these properties:

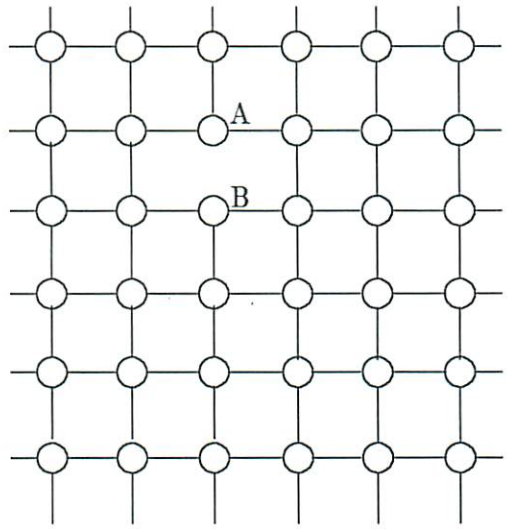
- The *cube-connected-cycles* topology proposed by Preparata and Vuillemin in [PV81]. This graph is based on a binary k-cube topology where each vertex is formed by a ring of nodes connected as a cycle, some nodes are also connected to nodes on neighboring vertices corresponding to a dimension of the binary cube. In the general case, the cube-connected cycles topology is **not** regular. In the examples we will consider a modified, but regular, version



a- Initial Condition



b- Node Failure



c- Link Failure

Figure 3.2: Node and Link Failures

of the cube-connected cycles where the number of nodes $N = k2^k$ and therefore each vertex consists of a ring with exactly k nodes each having 3 neighbors. The graph has a constant connectivity $n = 3$.

- The *torus* or wrap-around mesh such as the topology of the ILLIAC-IV multiprocessor system. The graph forms a surface where each node has a North, South, East and West neighbors. It is a regular topology with constant connectivity $n = 4$.
- The *binary k-cube* is a topology commonly known as the hypercube and used in several commercial systems [Sei85]. The number of nodes $N = 2^k$ where k is the *dimension* of the cube. Each node has k neighbor, one along each dimension of the cube. It is a variable connectivity graph with $n = \log_2 N$.

We define the following probabilities:

Definition 3.2 $P(i)$ = **Prob** *[the system is disconnected* exactly *after the i^{th} failure]*

Definition 3.3 $Q(i)$ = **Prob** *[a disconnected graph with $(N - i)$ nodes | a connected graph with $(N - i + 1)$ nodes and one node removal]*

Definition 3.4 $Q_K(i)$ = **Prob** *[a disconnected cluster of size K in a graph with $(N - i)$ nodes | a connected graph with $(N - i + 1)$ nodes and one node removal]*

Note that $Q(i)$ and $Q_K(i)$ are static conditional probabilities that do not take into account the evolution of the system. They are the probability of a disconnection event happening as the system goes from $(i-1)$ to i failures. $P(i)$, on the other hand, is a dynamic probability distribution function of the number of failures i. It implies that no disconnection occurred prior to the i^{th} failure. From these definitions, we can obtain the following relation:

$$P(i) = Q(i) \prod_{j=1}^{i-1} (1 - Q(j)) \tag{3.1.1}$$

This relation states that the probability of a disconnection event at the i^{th} failure is the product of the probability of no disconnection prior to that failure and the conditional probability of a disconnection at exactly the i^{th} failure. $Q(i)$ can also be written as:

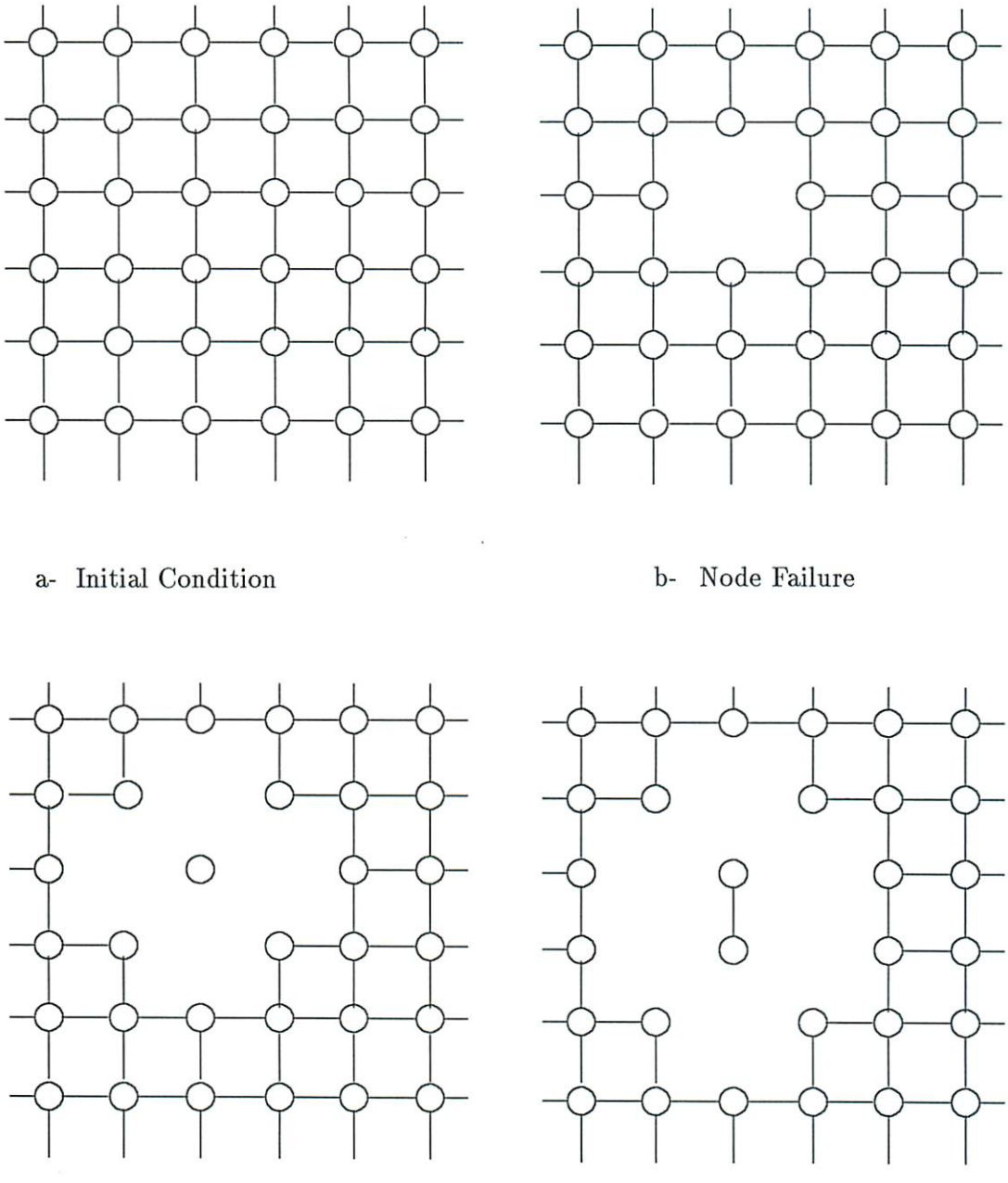
$$Q(i) = \sum_{K=1}^{M} Q_K(i) \tag{3.1.2}$$

Where M is the maximum possible value of K in a given graph topology.

3.1.2 Initial Disconnection Probability

A state of disconnection is depicted in Figure 3.3 for $K = 1$ and $K = 2$ in a torus topology. When a node fails (3.3b), its corresponding links are also lost. Multiple failures result in the disconnection of a cluster of nodes (3.3c and d).

Addressing the problem of the disconnection probability in the general case, for all possible values of K, is practically impossible. The number of possible combinations of K connected nodes



a- Initial Condition

b- Node Failure

c- Single Node Disconnection

$(K = 1)$

d- Two Nodes Disconnection

$(K = 2)$

Figure 3.3: Node failures and disconnection

in a given network topology can grow exponentially with K. We therefore choose to address first a simpler version of the problem, that is the probability of disconnecting a *single* node in the system.

Since in a regular topology every node has exactly n neighbors, we can state that:

$$P(i) = 0 \qquad for \;\; i < n$$

and

$$P(n) = Q_1(n)$$

For a single node to be disconnected, at least n nodes must fail. For the disconnection to occur at the $i = n$ failure, all the n neighboring nodes must fail. In a system with N nodes, there are $\binom{N}{n}$ ways this can happen, among N possible nodes to be disconnected. Therefore:

$$Q_1(n) = \frac{N}{\binom{N}{n}} \tag{3.1.3}$$

The above result can be extended to the conditional probability of disconnection for K nodes cluster and for $i = V_K$, as:

$$Q_K(V_K) = \frac{S_K}{\binom{N}{V_k}} \tag{3.1.4}$$

Conjecture 3.1 *For large N and $N >> n$, $Q_1(i) >> (Q(i) - Q_K(i))$ or $Q(i) \approx Q_1(i)$*

This conjecture states that in large networks, the single node disconnection event ($K = 1$) is much more probable than any K-cluster disconnection where $K > 1$. This conjecture can be explained intuitively by considering that as the cluster size K increases, the number of neighbors increases too. Therefore the likelihood of that many nodes failing in a pattern that would result in a disconnected cluster decreases.

Proof: The conjecture can be proved by an analysis of Equations 3.1.4 and 3.1.3. For large networks and small n, *i.e.* $N \gg n$, the following approximation is justified:

$$\binom{N}{n} \approx \frac{N^n}{n!}$$

Since V_K is $O(n)$ and S_K is $O(N)$, therefore:

$$\frac{Q_1(n)}{Q_K(V_K)} \approx \frac{N^{(V_K - n)}}{(V_K - n)!}$$

From Sterling's formula we can derive the following approximation:

$$m! \approx \frac{m^m}{e^m}$$

it follows that:

$$\frac{Q_1(n)}{Q_K(V_K)} \approx (\frac{Ne}{(V_K - n)})^{(V_K - n)}$$

In Section 3.4 we demonstrate that $Q_1(i+1) > Q_1(i)$ for $i > n$. It follows that:

$$Q_1(V_K) >> Q_K(V_K) \quad for \quad N >> n$$

Since we are considering failures occurring in sequence, this means that the probability of a single node disconnection at any $i < V_K$ is much larger than that of a cluster of size K at $i = V_K$. Therefore the single node disconnection probability is the dominant factor in $Q(i)$ and $Q(i) \approx Q_1(i)$.□

Following are two practical examples that demonstrate the above reasoning:

Example 1: Consider a torus topology ($n = 4$) with N nodes and the case of two nodes clusters ($K = 2$). In a torus, the number of neighbors to a two nodes cluster is $V_2 = 6$ and there are $S_2 = 2N$ such clusters in the network. Therefore the conditional probabilities of a single node disconnection at $i = n = 4$ failures ($Q_1(4)$) and of a two nodes disconnection at $i = V_2 = 6$ ($Q_2(6)$) are given respectively as:

$$Q_1(4) = \frac{N}{\binom{N}{4}} = \frac{24}{(N-1)(N-2)(N-3)}$$

$$Q_2(6) = \frac{2N}{\binom{N}{6}} = \frac{1440}{(N-1)(N-2)(N-3)(N-4)(N-5)}$$

For a system size of $N = 256$ nodes, we have:

$$Q_1(4) = 1054\ Q_2(6)$$

In section 3.1.4 we prove that $Q_1(i+1) > Q_1(i)$ for $i > n$, therefore:

$$Q_1(6) > 1054\ Q_2(6)$$

Example 2: A hypercube topology with $N = 2^n$ nodes, and $n = 8$. $S_2 = nN/2 = 2048$ and $V_K = (n - log_2 K)K$ therefore $V_2 = 14$.

$$Q_1(8) = \frac{256}{\binom{256}{8}} = 625\ 10^{-15}$$

$$Q_2(14) = \frac{2048}{\binom{256}{8}} = 0.2\ 10^{-18}$$

Hence:

$$Q_1(14) > 3\ 10^6\ Q_2(14)$$

These examples show that when the disconnection of a two nodes cluster is possible (at $i = 2(n-1)$) the probability of a prior single node disconnection event is a thousand times larger in the torus case and three million times larger in the hypercube case. The above two examples are not a proof, but a demonstration of the rationale behind the proposed conjecture for two common network topologies. In the following section an experimental approach based on a probabilistic, Monte-Carlo, simulation is presented that verifies the proposed conjecture.

	Cube-Connected Cycles, N=					
K	24	64	160	384	896	2048
1	**25.2**	**52.2**	**71.3**	**78.6**	**84.0**	**86.1**
2	18.6	13.0	15.3	14.96	12.2	12.0
3	11.0	10.0	5.0	3.78	2.5	1.5
4	7.0	6.5	3.0	1.22	0.9	0.4
	Torus, N=					
K	16	64	100	256	400	1024
1	**69.93**	**66.37**	**69.9**	**81.05**	**83.8**	**90.85**
2	18.1	10.8	11.07	8.4	8.15	4.05
3	8.13	6.17	6.2	4.55	3.95	2.6
4	1.45	4.01	3.6	1.75	1.05	0.75
	Binary Cube, N=					
K	16	64	128	256	512	1024
1	**67.75**	**77.25**	**89.1**	**92.4**	**95.1**	**97.3**
2	11.25	9.4	6.65	5.4	4.1	2.4
3	8.05	3.2	1.85	0.95	0.6	0.3
4	5.95	2.15	0.65	0.45	0.2	0.0

Table 3.1: Frequencies of Disconnection

3.1.3 Monte-Carlo Simulation

The objective of the simulation is to measure the values of $P(i)$ for different values of N on the three example topologies described in section 3.1.1: the cube-connected cycles, the torus and the binary k-cube.

The Monte-Carlo simulation algorithm used to obtain these results is described in Figure 3.4.

We first present the frequency of occurrence of a disconnections of different sizes clusters for the three topologies under considerations. This measure is defined as follows:

Definition 3.5 $F_{graph}(K)$ = **Prob***[the size of the disconnected cluster is* K | *a disconnection occurred]*

Given that a disconnection occurred in a given *graph*, $F_{graph}(K)$ is the probability that the disconnected component is a cluster of size K. F_t, F_{bc} and F_{ccc} are the values of F_{graph} for the torus, binary cube and cube-connected cycles topologies respectively. These values are shown in Table 3.1 for $K = 1, 2, 3$ *and* 4, as obtained from the Monte-Carlo simulation.

From these results, we can draw the following conclusions:

- Except for the cube-connected cycles with $N = 24$, in all three cases, and for all values of N, the proposed conjecture is verified in that the frequency of a single node disconnection, $F_{graph}(1)$, is larger than 50%. Furthermore, $F_{graph}(1)$ increases with increasing N independently of the connectivity.

- Comparing the results for the three topologies, one can observe that the dominance of the single node disconnection increases with an increasing n. For example, for $N = 64$, one can observe that: $F_{bc}(1) > F_t(1) > F_{ccc}(1)$ and $F_{bc}(K) < F_t(K) < F_{ccc}(K)$ for all $K > 1$

These effects can be explained by recalling the expression for $Q_K(V_K)$. Although the numerator, S_K, increases with an increased connectivity n, the increase in the denominator, $\binom{N}{V_K}$, is the dominant factor. Therefore, for large system sizes, the dominance of the single node disconnection probability on the overall disconnection probability can be stated by the following approximation:

$$Q_1(i) >> Q(i) - Q_1(i) \Longrightarrow Q(i) \approx Q_1(i)$$

which states that the conditional disconnection probability can be approximated by the single node disconnection probability. The above results do not give an indication on the value of $P(i)$ itself, only on its composition. In the following section we propose an approximation to $P(i)$ based on $Q(i) \approx Q_1(i)$ for large values of N.

3.1.4 Single-Node Disconnection Approximation

In this section we propose an approximate expression for $P(i)$ based on the results shown in the previous section. In this effort, the objective is **not** to provide an exact value for $P(i)$, but rather an indication of its order of magnitude in an analytical way.

The proposed approximation, for large N, is expressed as follows:

$$Q_1(i) >> Q(i) - Q_1(i) \quad or \quad Q_1(i) >> Q_K(i) \quad \forall K > 1 \tag{3.1.5}$$

Therefore we can approximate $P(i)$ as:

$$P(i) \approx P_1(i) = Q_1(i) \prod_{j=1}^{i-1} (1 - Q_1(j)) \tag{3.1.6}$$

In order to evaluate $P(i)$ we need an expression for $Q_1(i > n)$. It is provided by the following theorem:

Theorem 3.1 *The conditional probability of disconnecting a single node after i failures, where $n < i < 2n - 1$ is given by:*

$$Q_1(i) = \frac{nN \binom{N-n-1}{i-n}}{(N-i+1)\binom{N}{i-1}} \; for \; n < i < 2n-1; \tag{3.1.7}$$

Proof: Keeping in mind that no disconnection happened at the $(i-1)^{st}$ failure, the probability that one occurs at the i^{th} failure is the probability that some node had all but one of its neighbors failed *and* that that neighbor was the i^{th} failure. $\binom{N}{i-1}$ represents the possible combinations of $i-1$ failures among N nodes. $\binom{n}{n-1} = n$ is the possible combinations of $n-1$ failed neighbors among n neighbors. $\binom{N-n-1}{i-n}$ is the combination of the remaining $i-n$ failures

```
for (j = 0; j < Number-of-iterations; j++) {
        Build a graph;
        for (i = 1; i < N; i++){
                • choose a node at random from the remaining (N − i);
                • remove that node and all its links from the graph;
                • traverse the graph recording the number
                        and size of connected components;
                • if (disconnection){
                        record i;
                        record size of cluster;
                        exit;}
        Report data;}
}
```

Figure 3.4: Monte-Carlo Simulation Algorithm

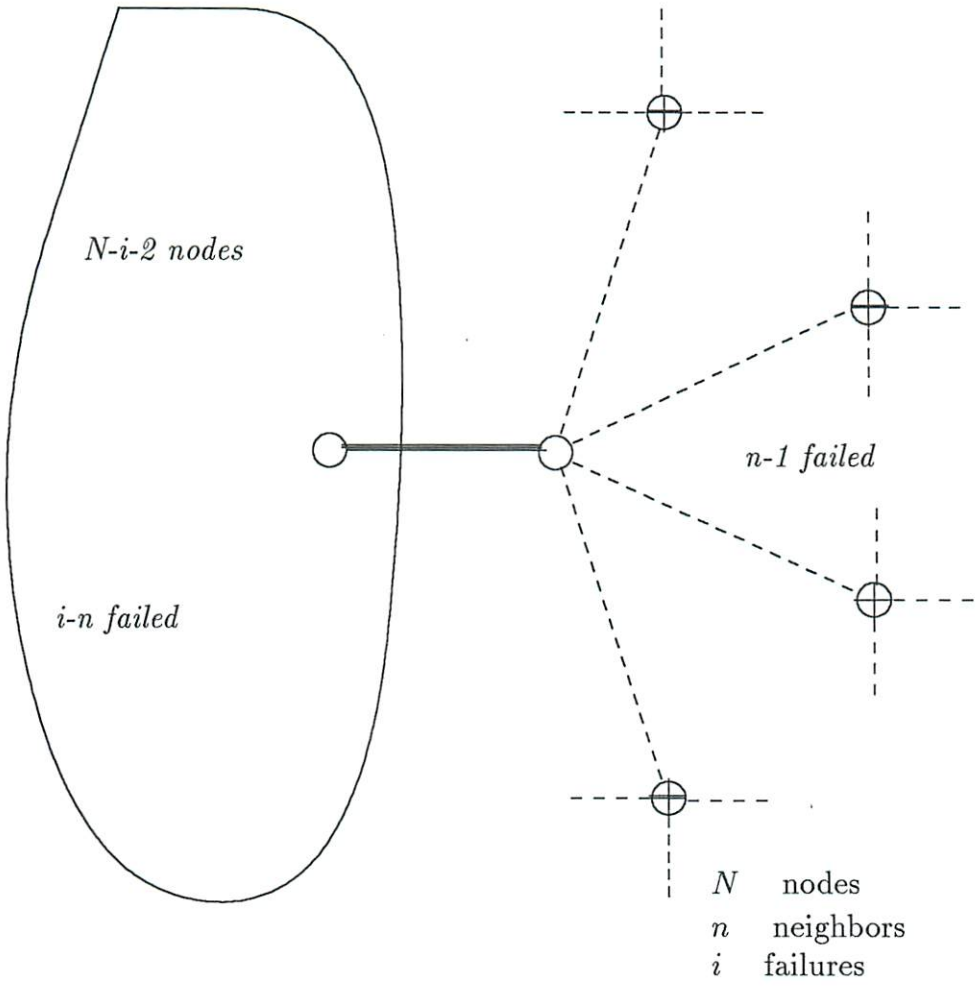


Figure 3.5: Disconnection when $i > n$

in the rest of the system, where N is the number of nodes which can be isolated, and $\frac{1}{N-i+1}$ is the probability that the last remaining neighbor fails. □ Equation 3.1.7 corresponds to the single node disconnection probability when more than n nodes have failed. For $i \geq 2n-1$ it is possible to have two ore more single node disconnections. However, the probability of multiple single node disconnections is of the same order as that of a cluster disconnection where $K > 1$. Therefore, using the approximation of Equation 3.1.5, we can extend the range of i in Equation 3.1.7 to $i > n$. Simplifying Equation 3.1.7 we obtain:

$$Q_1(i > n) = \frac{n(N-n-1)!(i-1)!(N-i)}{(i-n)!(N-1)!} \tag{3.1.8}$$

From Equation 3.1.8 we can derive:

$$\frac{Q_1(i+1)}{Q_1(i)} \frac{i}{i+1-n} \frac{N-i-1}{N-i} \approx \frac{i}{i+1-n}$$

This proves the relation $Q_1(i+1) > Q_1(i)$ for $i > n$.

The single node disconnection approximation can therefore be summarized as:

$$P(i) = 0 \quad for \;\; i < n$$

$$P(n) = Q_1(n) = \frac{N}{\binom{N}{n}}$$

$$P(i) = Q(i) \prod_{j=1}^{i-1} (1 - Q(j))$$

and

$$Q(i) \approx Q_1(i) = \frac{n(N-n-1)!}{(N-1)!} \frac{(i-n)!}{(i-1)!(N-i)} \quad for \;\; i > n$$

In the following section we present a comparison of the disconnection probability values obtained with this approximate analytical model and the Monte-Carlo simulation described in the previous section.

3.1.5 Analytical and Simulation Results

Using the same simulation approach described in Section 3.3, we obtained the values of $P(i)$ for the three topologies and for different values of N.

Figure 3.7 shows the values of $P(i)$ for a torus with 64 nodes as a function of i. The two curves show the simulation and analytical values. The maximum difference in value, at the peak, is 25%. Figure 3.7 shows the same curve for the case of a binary cube, the maximum difference being 20%. Figures 3.9 and 3.8 show similar plots for $N = 256$. One can notice that the curves for the binary cube cases are narrower and peak around $i = N/2$, while they are wider for the torus cases with a peak at $i < N/2$.

Table 3.2 summarizes the comparison between the simulation results and those obtained with the analytic approximation. These tables show the peak value of the overall disconnection probability distribution function $P(i)$, denoted by P_{max}, and the corresponding value of the number of node failures, i_{peak}. The objective in this comparison is to show the order of magnitude of $P(i)$ and the corresponding *range* of values of i where the peak value of the disconnection probability occurs.

From this table one can observe the following:

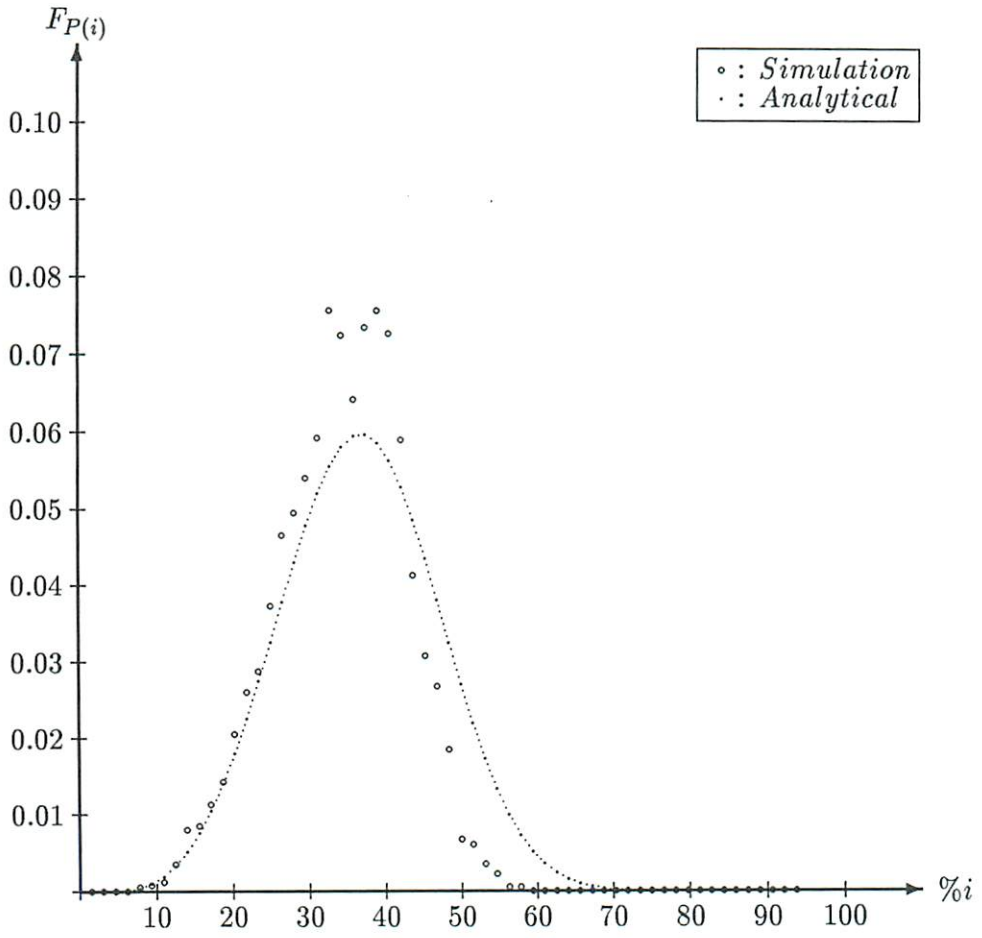


Figure 3.6: Probability of Disconnection, Torus, $N = 64$

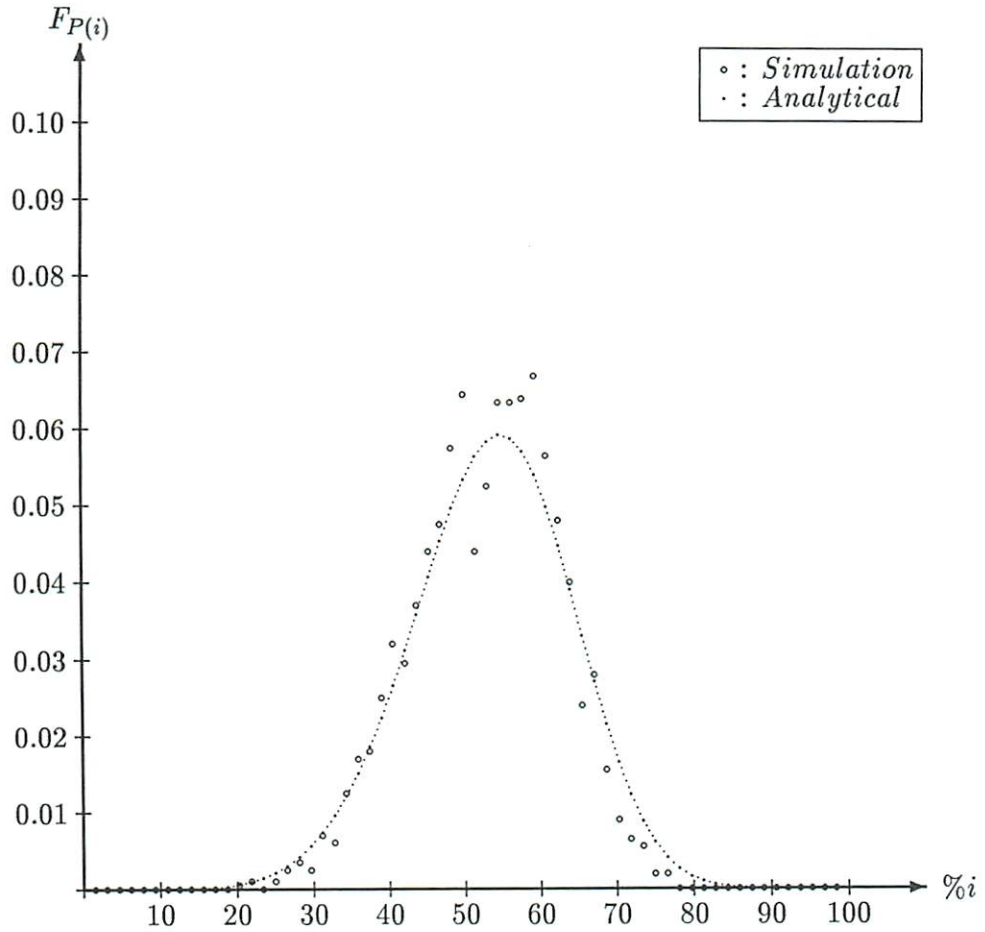


Figure 3.7: Probability of Disconnection, Binary Cube, $N = 64$

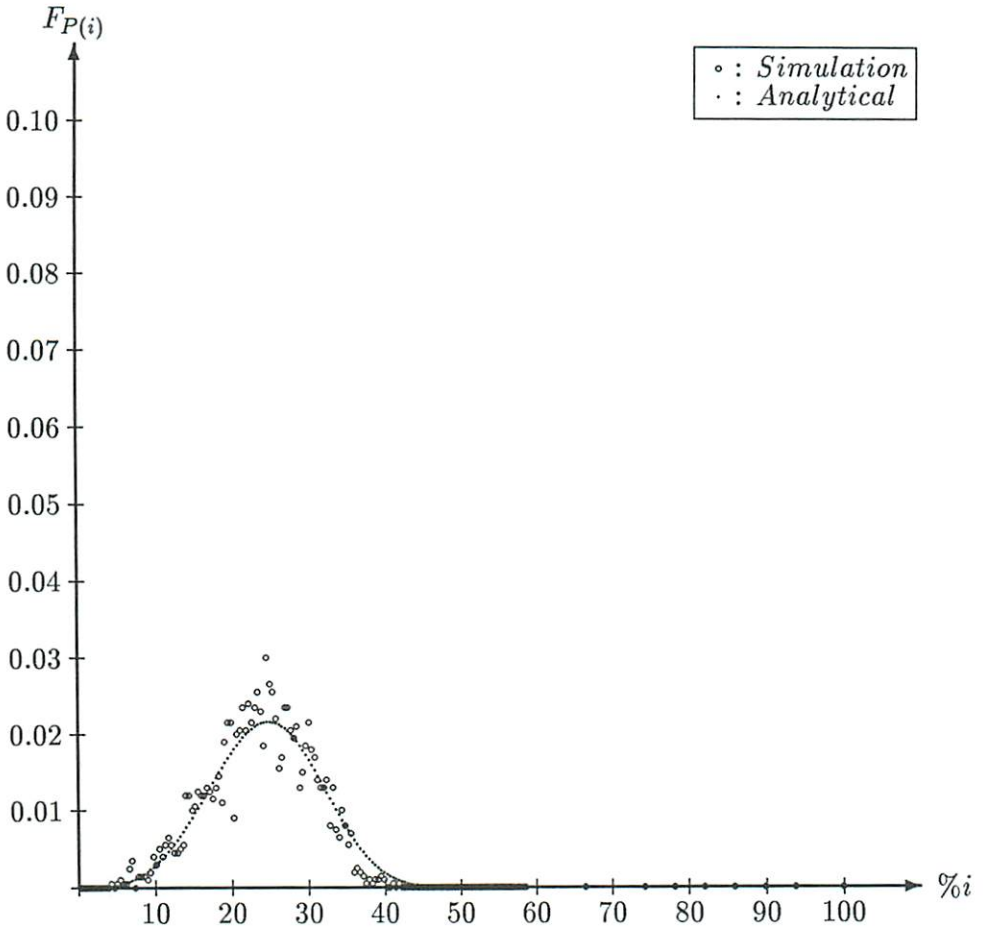


Figure 3.8: Probability of Disconnection, Torus, $N = 256$

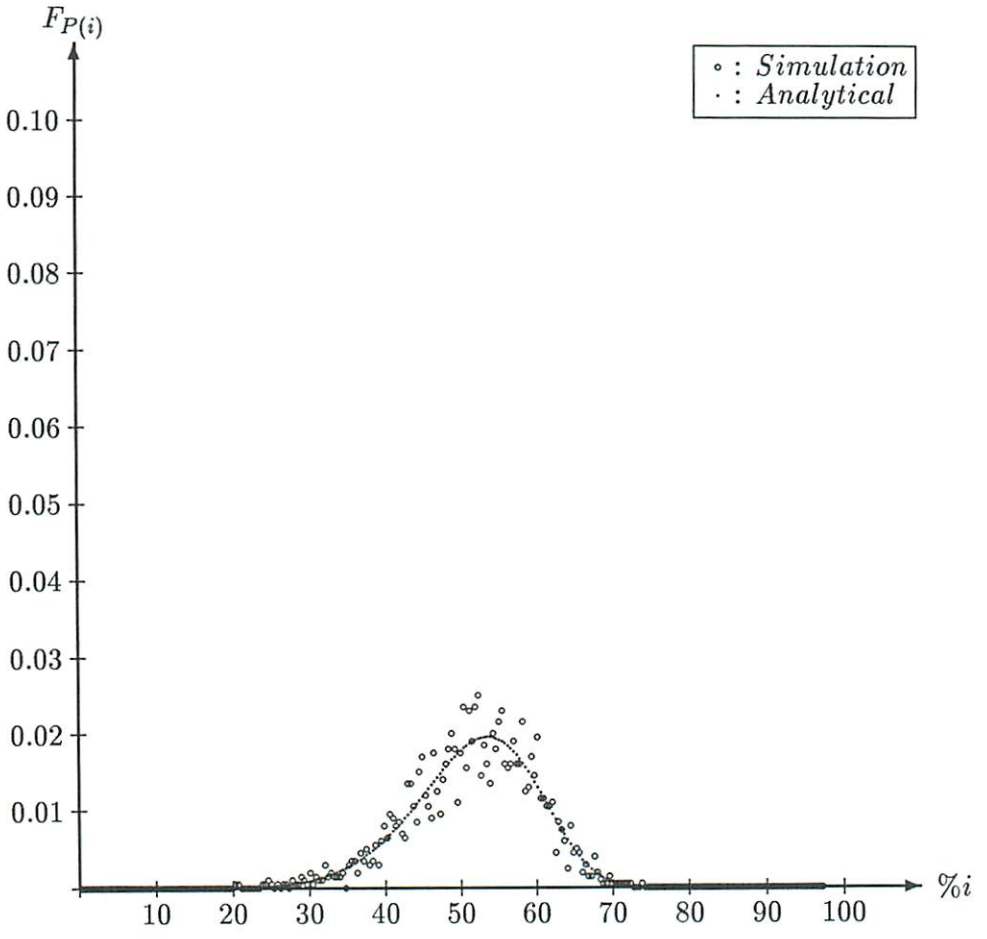


Figure 3.9: Probability of Disconnection, Binary Cube, $N = 256$

		Cube-Connected Cycles					
N		24	64	160	384	896	2048
Simulation	P_{max}	0.296	0.095	0.044	0.028	0.0187	0.016
	i_{peak}	6	15	28	50	84	114
Analytical	P_{max}	0.132	0.068	0.0375	0.021	0.012	0.007
	i_{peak}	9	16	28	48	84	144
		Torus					
N		16	64	100	256	400	1024
Simulation	P_{max}	0.2066	0.0775	0.058	0.03	0.0215	0.0105
	i_{peak}	9	25	32	63	88	190
Analytical	P_{max}	0.1765	0.0597	0.043	0.0216	0.0156	0.00785
	i_{peak}	9	24	32	64	88	175
		Binary Cube					
N		16	64	128	256	512	1024
Simulation	P_{max}	0.2122	0.069	0.0375	0.025	0.0165	0.0117
	i_{peak}	9	38	67	142	284	575
Analytical	P_{max}	0.1765	0.0593	0.0342	0.0195	0.0110	0.0062
	i_{peak}	9	35	69	137	272	541

Table 3.2: P_{max} and i_{peak}

- A very close correlation in the value of i_{peak} between the simulation results and the analytical model. The variation in the value of P_{max} is due to the approximation in the analytical model as well as to the statistical error in the simulation.
- In the cube-connected and mesh topologies, $i < N/2$ while $i_{peak} > N/2$ in a hypercube topology. For example, with $N = 1024$ we have $i \approx 200$ for the mesh and $i \approx 550$ for the hypercube. This shows that higher graph connectivity allows the network to sustain a larger number of failures before disconnection.
- The value of P_{max} falls within the same order of magnitude for all three topologies and for a given range of values of N. For example, $P_{max} \approx 20\%$ for $N = 16$ and $P_{max} \approx 2.5\%$ for $N = 256$.

These results show that the number of nodes in the system, N, is the dominant factor in determining the maximum value of the disconnection probability, P_{max}. On the other hand, the node connectivity, n, determine the range of values of i_{peak}. In other words: the *magnitude* of the disconnection probability id determined by the system size (N), while the expected number of failures that can be sustained before a disconnection occurs is determined by both the system size and the node connectivity.

The coverage factor, in a gracefully degradable system, is the probability of successful recovery. We have shown that a disconnection in the network will prevent a successful recovery mechanism. Therefore, the probability of *no* disconnection is a multiplicative coefficient in the expression of the coverage factor. In other words, the coverage factor at the i^{th} failure can be expressed as:

$$c_i = (1 - P(i))c_i'$$

	Cube-Connected Cycles					
N	24	64	160	384	896	2048
Simulation	2	4	7	12	20	40
Analytical	2	4	7	12	21	35
	Torus					
N	16	64	100	256	400	1024
Simulation	4	8	11	19	28	55
Analytical	4	8	11	21	30	59
	Binary Cube					
N	16	64	128	256	512	1024
Simulation	4	18	33	75	159	336
Analytical	4	17	36	77	161	337

Table 3.3: Network Resilience for $p = 0.01$

where c_i' is the coverage factor in a system where no disconnection can occur as for example in a fully connected graph topology. The range of values of P_{max} shown in Table 3.2 is very high compared to any acceptable value of the coverage factor. The question therefore becomes: What is the number of nodes that can be allowed to fail with a reasonably low probability of disconnection? This evaluation is the topic of the next section.

3.2 Network Resilience

Network Resilience is introduced as measure of the expected number of nodes failures a system graph can sustain with a *reasonable* probability of no network disconnection. The reasonable probability is determined by a *certainty factor* $(1 - p)$. Network resilience is therefore defined as:

$$NR(p) = \sum_{i=1}^{NR} P(i) \leq p \tag{3.2.9}$$

NR is therefore the cumulative distribution function of $P(i)$. In a similar fashion, we define the measure of *Relative Network Resilience*, RNR, as:

$$RNR(p) = \frac{NR(p)}{N} \tag{3.2.10}$$

While $NR(p)$ measure the absolute number of nodes, $RNR(p)$ measures the percentage of nodes and gives therefore an indication of the scalability of various network topology with respect to network fault-tolerance.

Figure 3.10 shows the plots of $NR(p)$ for all three topologies for $p = 0.01 = 1\%$. Figure 3.11 shows similar plots of $RNR(0.01)$. The numerical values are displayed in Table 3.2.

It is interesting to notice that for increasing N the values of $RNR(p)$ decreases in the case of the cube-connected cycles and mesh while it increases for the hypercube. For the range of values shown in Table 3.2 for the respective topologies, this ratio goes from 8% ($N = 24$) to 1.9% ($N = 2048$)

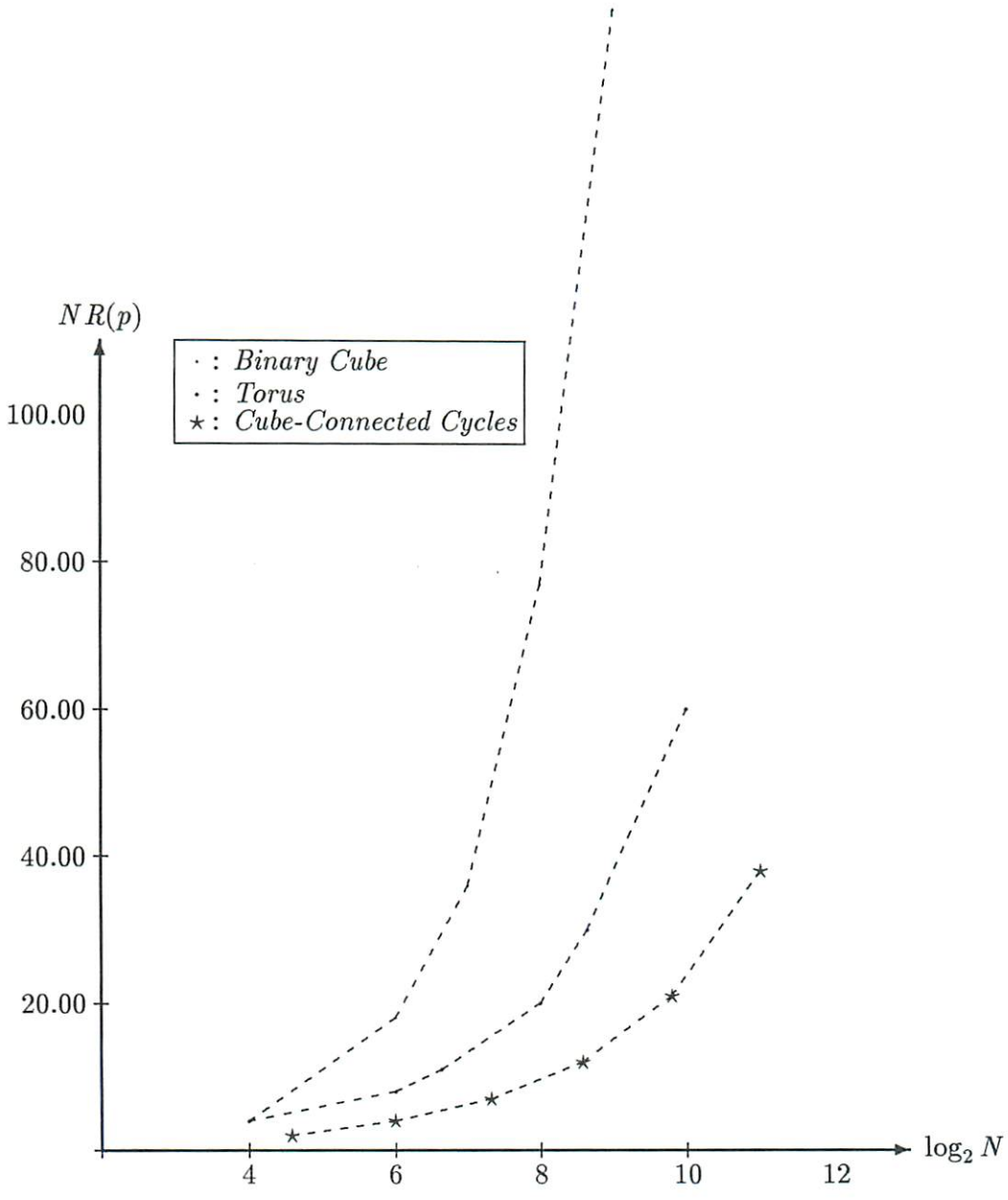


Figure 3.10: Network Resilience, $NR(p)$, for $p = 0.01$

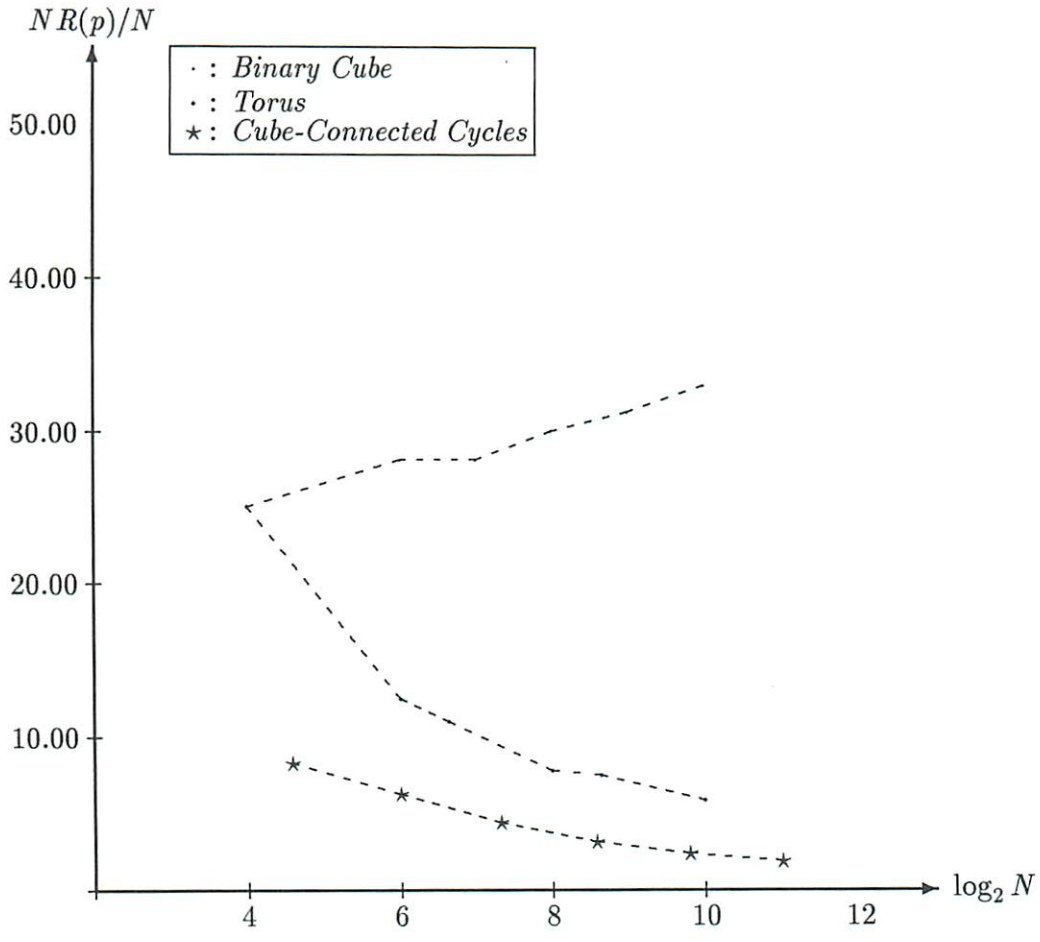


Figure 3.11: Relative Resilience, $NR(p)/N$, for $p = 0.01$

for the cube-connected cycles, from 25% ($N = 16$) to 5.6% ($N = 1024$) for the mesh and from 25% ($N = 16$) to 33% ($N = 1024$) for the hypercube.

From these observations we can conclude:

- The value of $NR(p)$ increases with increasing N. Therefore, larger systems can allow a larger number of nodes to fail before the cumulative probability of disconnection reaches a given level.
- The value of $RNR(p)$ decreases with increasing N for constant constant connectivity graphs such as the torus or the cube-connected-cycles. Therefore, for a constant n, the number of degradation states is a decreasing fraction of the number of nodes as N increases.
- For networks where the connectivity n is an increasing function of the number of nodes N, as in the binary cube case, an increase in N increases both $NR(p)$ and $RNR(p)$.

The comparison between analytical and simulation values in Table 3.2 indicates that the adopted analytical model yield very close values of $NR(p)$ and $RNR(p)$. While network fault-tolerance is a static measure that depends exclusively on n and therefore on the topology alone, network resilience is a probabilistic measure that depends on both N and n. It can be tuned by selecting appropriate values of the certainty factor p. In fact for the trivial and conservative case where $p = 0$ the two measures of network fault-tolerance, as defined in [Pra85a], and network resilience as defined here are equivalent.

The notion of network resilience has implications on the number of degradation states that can be allowed in a degradable system. A large-scale gracefully degradable system can tolerate element failures while providing continued operations. The maximum number of node failures allowed in a multicomputer system is called the *degradation level*, denoted here as D. When the application does not pose any constraint on the minimum number of processing nodes necessary, the value of D is determined by the system fault-tolerant design. As the number of failures increases, the difficulty of system recovery increases, thereby decreasing the probability of a successful recovery. The degradation level, D, is therefore the number of failures up to which graceful degradation can be expected with reasonably high probability. After D failures, the recovery becomes difficult and the probability of success low, therefor the system is considered failed. In a distributed system that is not based on a fully connected graph, the value of D is constrained by the probability of network disconnection. By determining an acceptable value of the certainty factor p, network resilience can allow the determination of the number of degradation levels D.

3.3 Non-Regular Graphs

Non-regular graph topologies such as the array, offer the advantages of a constant connectivity that is independent of the system size and therefore of node modularity. Furthermore, the array, like the binary k-cube, is a *scalable* topology: An array (or binary k-cube) graph can be partitioned into several smaller arrays (cubes) having the same topology. This is not the case in graphs such as the torus where partitioning does not preserve the original network topology.

In this section we present a simulation analysis of the disconnection probability in an array topology. These results are then compared to those of a same size torus using the analytical model that was derived in section 3.1.

Table 3.4 shows the frequency of disconnection of K nodes cluster as a function of the array size. These results show that the frequency of a single node disconnection ($K = 1$) is still the dominant

	Array					
K	16	64	100	256	400	1024
1	55.7	49.7	56.8	65.7	72.6	76.0
2	12.6	10.9	10.9	9.9	9.9	15.0
3	13.5	10.4	8.7	7.7	5.8	3.0
4	8.6	5.2	6.1	3.7	3.8	3.0

Table 3.4: Frequencies of Disconnection, Array

N	16	64	100	256	400	1024
Torus	4	8	11	21	30	59
Array	1	2	3	10	16	23

Table 3.5: Comparison of Network Resilience, Torus and Array ($p = 0.01$)

factor. However, comparing the frequencies of disconnection in the array and torus topologies (Table 3.1), we can note that the frequency of single node disconnections in the array case is significantly smaller than in the torus case. This reduction is due to the *edge effect* introduced by the non-regularity of the graph. In an array topologies there are $4(\sqrt{N} - 1)$ nodes with degree 3 and 4 nodes with degree 2. The ratio of the number of edge nodes to the total number of nodes is $O(N^{\frac{1}{2}})$. It would be expected, therefore, that for very large system sizes the disconnection frequency in an array topology would approach that of a torus topology.

We note from Figure 3.12 that the edge effect results in an increase in the probability of disconnection as compared to that of a torus. Therefore, for a same fraction of failed nodes, the array is more likely to get disconnected than a torus.

Table 3.3 shows the effects of graph non-regularity on Network Resilience by comparing $NR(0.01)$ for the array and the torus graphs. These values show a very large decrease in network resilience. In fact, one can notice that for $N > 100$ the NR of an array is roughly half that of a same size torus. Therefore an array topology is twice as prone to network disconnection than a torus of the same size.

From this analysis, we can conclude that the problem of network disconnection is even more acute in the case of low degree non-regular graphs such as the array. Therefore, protection measures, such as backup nodes or links, must be provided in order to allow a larger number of graceful degradation states in systems based on such graph topologies. An alternative possibility would be to investigate the design of network topologies that minimize the ratio of edge nodes to total number of nodes thereby providing a higher resilience to network disconnection.

3.4 Link Failures

The analysis of sections 3.1 and 3.3 are based on the assumptions of node failures exclusively. In this section, we carry a similar analysis for the link failures, assuming no node failures.

In the multicomputer system model described in section 3.1, each node consists of a Processing Unit (PU) and a Communication Unit (CU). The failure of either of the PU or CU is tantamount to a node failure since no communication is possible to the outside world (in the case of a CU failure)

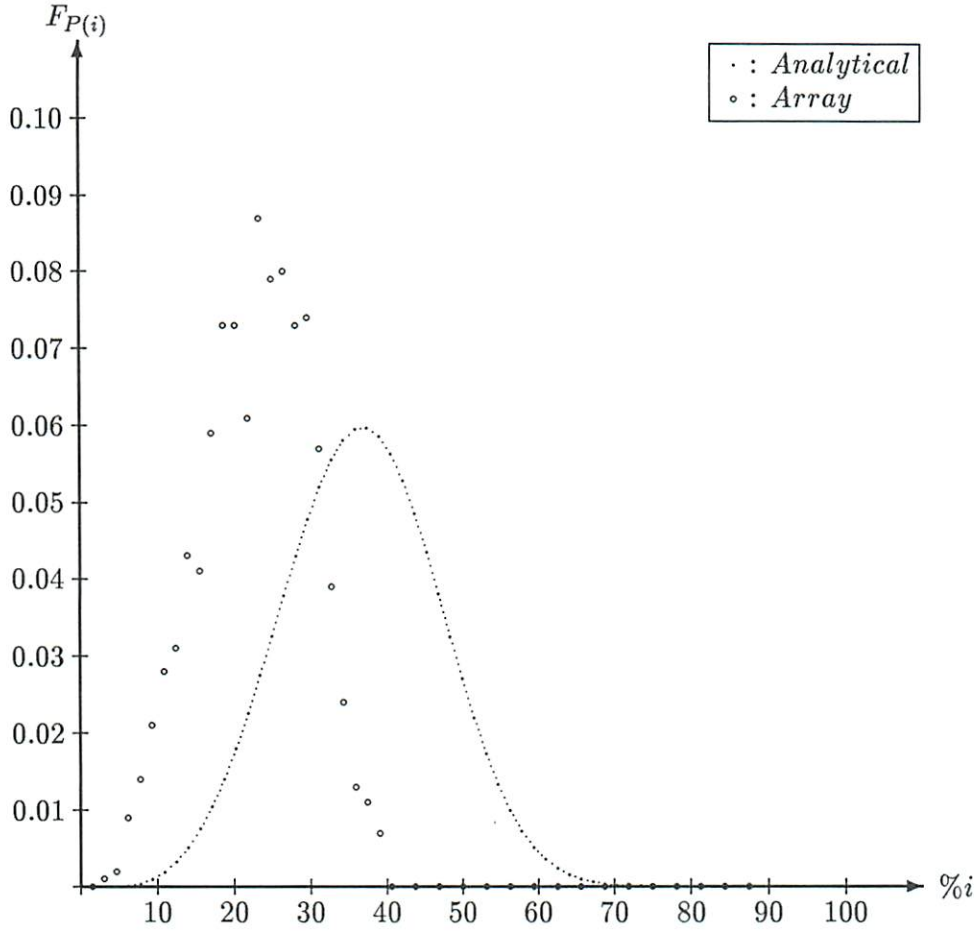


Figure 3.12: Probability of Disconnection, Array and Torus, $N = 64$

	Cube Size					
K	8	16	64	128	256	512
1	76.3	79.3	87.7	90.3	95.3	97.0
2	18.6	15.3	9.3	7.8	4.6	2.6
3	4.2	3.6	1.8	1.7	0.1	0.4
4	0.9	1.4	0.8	0.1	0.0	0.0

Table 3.6: Frequencies of Disconnection in a Binary Cube, Link Failures

or no useful work can be done in that node (in the case of a PU failure). The links connecting the nodes constitute the interconnection network.

Well known and simple fault-tolerance techniques exist for the protection against link failures. Most common among these are the various error detection and correction schemes. These are based either on the use of redundant bits for error-detection and error correction such as in SEC/DED codes, or error-detection bits with data retransmission (SED codes). Although these schemes can provide a very high level of protection, they are not fail-proof and therefore system failures can still be caused by single or multiple link failures. A particular case worth noting is when Very Large or Wafer Scale Integration technologies are used to implement several nodes on one chip. In such cases the silicon real-estate used by the interconnection network becomes a substantial fraction of the overall chip area. However, the same traditional protection techniques apply as well. Therefore, in all realistic cases, the expected link failure rate will be much lower than the expected node failure rate.

A Monte-Carlo simulation of link failure in binary n-cubes (Table 3.6) shows that the same behavior can be observed as in the node failure case. The most frequent disconnection is that of a single node. We can deduce, therefore, that for either link or node failures, protecting against the disconnection of a single node would provide a very high coverage.

Chapter 4

COMPUTATIONAL RELIABILITY

"... but I hope this restores your confidence in my reliability."

"I am sorry about this misunderstanding, Hal," replied Bowman, rather contritely.

Arthur C. Clarke
2001: A Space Odyssey

In Chapter 3 we have seen that an increase in system size has a favorable effect on the network disconnection probability by allowing a higher value of the Network Resilience. For low constant connectivity graphs, this value was shown to be a decreasing fraction of the system size as the number of nodes is increased. As the system size is increased, the performance and reliability of the system are influenced, respectively, by (1) an increase in the available computational power and (2) an increase in the expected rate of failure of the system. The objective of this chapter, therefore, is to propose a combined performance/reliability modeling and an analysis of gracefully degradable large-scale multicomputer systems. In this analysis, particular emphasis will be put on the following issues:

- the *scalability* of large-scale systems, by analyzing the effects of an increase in system size;
- the *quality of the recovery scheme*, by an analysis of the effects of the coverage factor.
- the *computational reliability* which is the probability of correct completion of a given computation, as function of both the system size and the coverage factor.

We present first the system and failure models on which the analysis in this chapter is based. This followed by an analysis of time-based measures such as Mean-Time-To-Failure and Mission-Time. Computation-based measures are then introduced, defined and analyzed. Finally, a discussion of these results and their implications on the design and performability of LSCSs is given.

4.1 System and Fault Models

The model under consideration is that of a large-scale, homogeneous multiprocessor. The computation is, initially, uniformly partitioned among N identical processing elements. The system is assumed to support graceful degradation. Upon the detected failure of a processor, its computational load is picked up by another processor or set of processors with near uniform load partitioning. Distributed fault-tolerant schemes are based on the algorithm proposed by Preparata

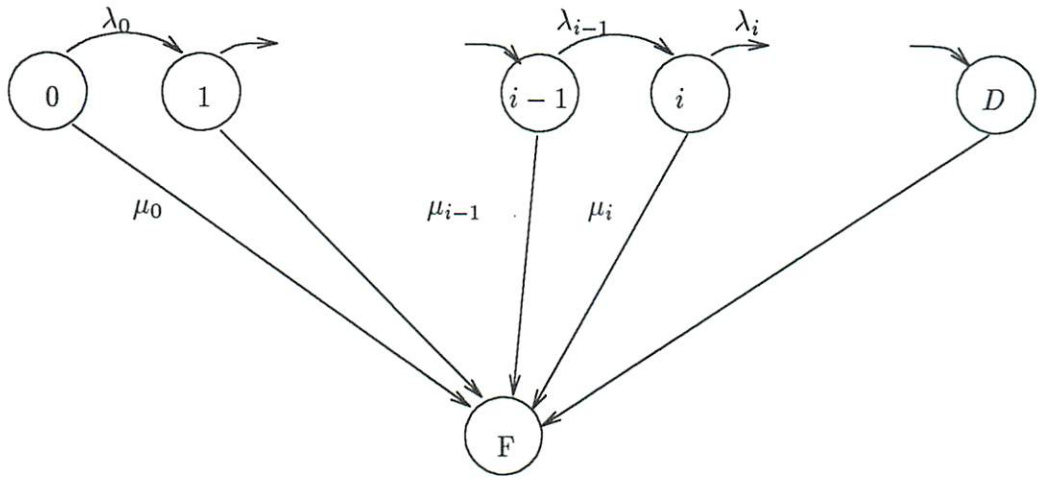


Figure 4.1: Markov model of failures

et al. [PMC67]. A detailed description and discussion of this algorithm can be found in [KR80]. It is based on the following three steps:

- fault detection
- fault isolation
- system reconfiguration and recovery

However, the ability of a system to gracefully degrade hinges on the *combined* success of these three steps. The failure to either detect, isolate or recover from a fault can result in a total system failure. The cumulative probability of success of these three steps is expressed by the *coverage factor* [BCJ*71]. In a distributed system, the recovery procedure relies on the communication among processors, and therefore on the system network being connected. Unless the system has a fully connected network topology, successive failures might result in a partitioning of the network. The probability of disconnection is, therefore, a parameter of the coverage factor as demonstrated in Section 3.1.2.

A common assumption of fault-tolerant schemes is that of no simultaneous multiple failures, in other words, that failures are sufficiently spaced to allow the recovery process to deal with one failure at a time. The occurrence of two or more failures in a short interval might lead to a failed recovery and therefore to a total system crash. For this reason, a *Global State Saving* (GSS) procedure is necessary at fixed intervals during the course of the computation to insure that the progress done by the computation is not lost. One of the goals of the present analysis is to evaluate the interval between successive GSS procedures: T_{gss}.

For the sake of simplicity, the analysis that follows will not take into consideration the time overhead incurred in recovering from a failure. Although this assumption is unrealistic, it is justifiable in an analysis of asymptotic behavior.

The system is modeled by a continuous-time Markov chain (CTMC), shown in Figure 4.1, [Tri82] [SS82]. Since our analysis will focus on gracefully degradable systems, we will not consider system repair and therefore the CTMC is acyclic. The following parameters are used:

- $P_i(t)$ is the occupation probability of *state* i, i being the number of failed processors, $i = 0, \ldots, D-1, F$; where F is the state of total system failure.

- D is the number of allowable degradation states, expressed as a function of N.
- c_i is the a state-dependent coverage factor, which is the probability of successful recovery from a single failure in state i.

The rates of state transitions, λ_i and μ_i, can be expressed as a function of the single processor failure rate λ, as follows:

$$\lambda_i = c_i(N-i)\lambda$$

$$\mu_i = (1 - c_i)(N-i)\lambda$$

In the analysis which follows we will assume, for simplicity, a constant (*i.e.,* state independent) coverage factor.

$$c_i = c \qquad i = 0, \ldots, D-1$$

This assumption is somehow unrealistic since the probability of partitioning the network increases with increasing i and might reach values comparable to any assumed value of c (see Chapter 3).

Let $P_i(t)$ be the state occupancy probability of state i; in other words it is the probability of the system having exactly i failures. The steady-state solution of this Markov model is described by the following differential equations:

$$\begin{aligned}
\frac{dP_0(t)}{dt} &= -(\lambda_0 + \mu_0)P_0(t) = N\lambda P_0(t) \\
\frac{dP_i(t)}{dt} &= -(\lambda_i + \mu_i)P_i(t) + c\lambda_{i-1}P_{i-1}(t) \\
&= -(N-i)\lambda P_i(t) + c(N-i+1)\lambda P_{i-1}(t) \qquad i = 1, \ldots, D-1 \\
\frac{dP_F(t)}{dt} &= \lambda P_D(t) + \lambda(1-c)\sum_{j=0}^{D-1}(N-j)P_j(t)
\end{aligned}$$

Subject to the following constraints:

$$\begin{aligned}
P_0(0) &= 1 \\
P_i(0) &= 0 \qquad i = 1, \ldots, D-1
\end{aligned}$$

The state probabilities can be derived as:

$$\begin{aligned}
P_0(t) &= e^{-N\lambda t} \\
P_i(t) &= c^i \begin{pmatrix} N \\ N-i \end{pmatrix} (e^{-\lambda t})^{(N-i)}(1 - e^{-\lambda t})^i \qquad i = 1, \ldots, D-1
\end{aligned}$$

The reliability $R(t)$ is simply the probability of being in any one of the states $i = 0, \ldots, D-1$.

$$R(t) = \sum_{i=0}^{D-1} P_i(t) \tag{4.1.11}$$

The *mean time to failure* ($MTTF$) which is the expected time to first failure, is:

$$MTTF = \int_0^\infty R(t)dt \tag{4.1.12}$$

The Mission Time, MT, is defined for a given minimum reliability R_{min} as:

$$R(MT) = R_{min} \tag{4.1.13}$$

We define $F_P(t)$ to be the expected number of failed processors at time t:

$$F_P(t) = \sum_{i=0}^{D-1} iP_i(t) \tag{4.1.14}$$

Unless otherwise noted, in the rest of this discussion, we will assume a fully degradable system. This means that the system allows graceful degradation for up to $N-1$ failures, in other words, $D = N-1$. The unit-time will be taken as $1/\lambda = MTTF_1$ (i.e, the MTTF of a single processor) and a value of $R_{min} = 0.99$.

4.2 Time-Based Reliability Analysis

In this section, we present a reliability analysis of large-scale degradable systems based on two time measures: (1) the Mean-Time-To-Failure ($MTTF$) and (2) the Mission-Time (MT). The Mission Time measure is primarily intended to evaluate the reliability of mission-oriented applications such as non-repairable, on-board systems. It is used in this analysis as a measure of the time interval where $R(t) \geq R_{min}$.

In both cases we will use the results to evaluate the interval T_{gss} assuming it can be expressed as a function or a fraction of the $MTTF$ or the MT respectively.

4.2.1 MTTF-based evaluation

The expression for $MTTF$ can be obtained from Equations 4.1.12 and 4.1.11 as:

$$MTTF = \int_0^\infty \sum_{i=0}^{D-1} P_i(t)dt = \frac{1}{\lambda c} \sum_{i=1}^{D-1} \frac{c^i}{i} \tag{4.2.15}$$

The expression for $MTTF$ indicates that increases in N have diminishing effects on the value of the expected time to first failure. In fact, an increase from N to $N+1$ processors results in a minimal increase in $MTTF$:

$$MTTF(N+1) - MTTF(N) = \frac{c^{N+1}}{N+1}$$

since $c < 1$, the increase becomes insignificant for large values of N.

The values of $MTTF$ are plotted in Figure 4.2 as a function of N for different values of c. The series in Equation 4.2.15 is not convergent but has a logarithmic behavior. Therefore there is no asymptotic limit to $MTTF$. However, for all practical purposes, the mean time to failure can be considered constant for sufficiently large N given a value of the coverage factor.
Let N_k be the value of N at the knee of the curve in Figure 4.2, i.e

$$MTTF(c, N) \approx MTTF(c, N_k) \qquad for \;\; N > N_k$$

An analysis of the values in Figure 4.2 shows the following:

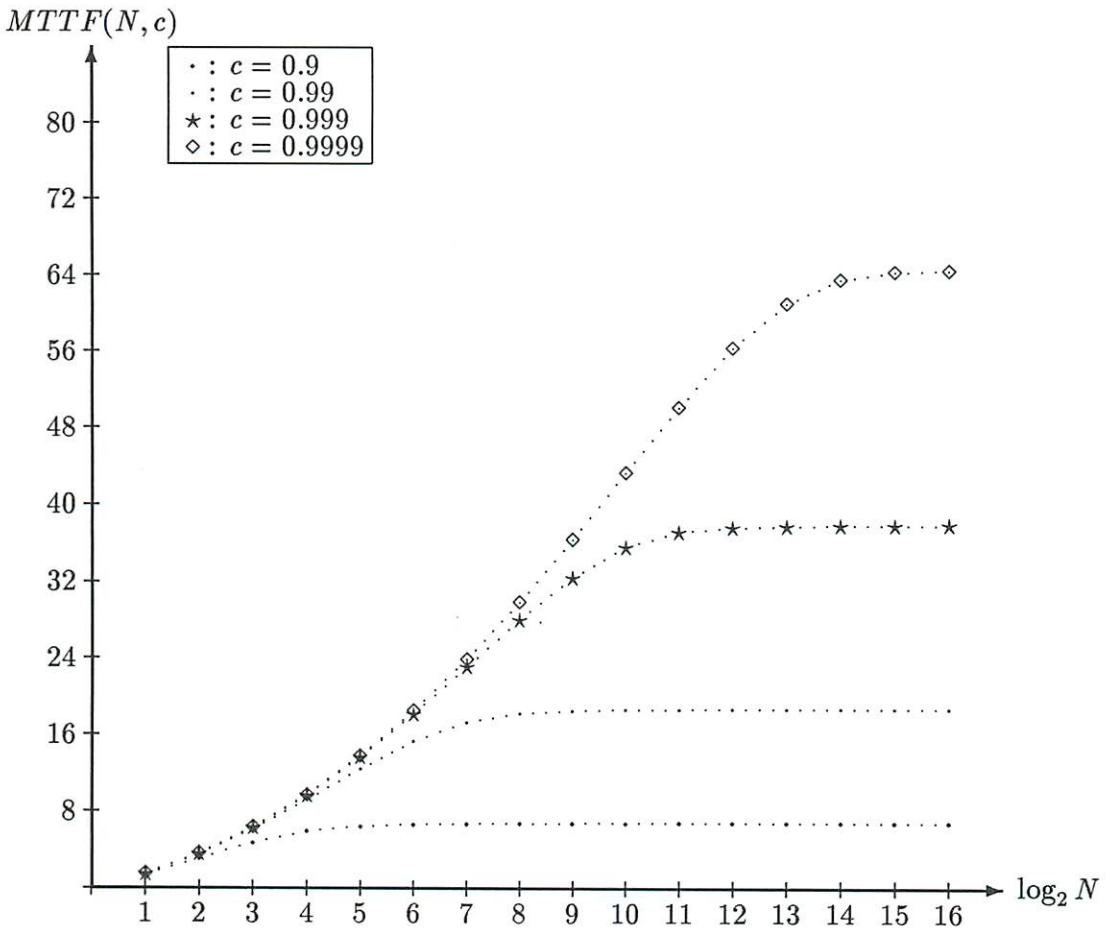


Figure 4.2: $MTTF$ as function of N and c

- The effects of a larger N on the $MTTF(N, c)$ are significant compared to the MTTF of a single processor. For example, for $c = 0.99$ and $N = 128$ the expected time to first failure is 16 times that of a single processor, or $MTTF(128, 0.99) = 16\ MTTF(1)$.

- For smaller values of N the coverage c has little effect on the values of the MTTF. For example, for $N = 32$, the expected time to first failure $MTTF(32, c) \approx 13$ for all values of $c \geq 0.99$.

- As the probability of failed recovery, $(1 - c)$, is decreased by a factor of 10 the corresponding increase in $MTTF$ is ≈ 2. For example , for $N > 128$ and an increase in c from 0.9 to 0.99 results in an increase in MTTF from 6.8 to 18.6 (factor of 2.7). This means that substantially large improvement on the reliability of the recovery procedure do not significantly affect the expected time to failure.

From the above analysis, we can conclude that:

1. For smaller systems, the probability of successful recovery has very little effect on the mean time to first failure.

2. For larger systems, the mean time to first failure is a constant function of the coverage factor and is independent of the number of processors.

An $MTTF$ based evaluation of T_{gss} implies that for a given value of c and $N > N_k$, the interval can be kept constant independently of the number of processors. However, as N increases, the amount of *computational work* performed during that interval increases and the computation progresses faster.

A drawback of an $MTTF$ based evaluation is that it cannot take into account the overall system reliability. Of particular is the system reliability at the time when the global state saving procedure is performed. In fact, if $R(T_{gss})$ is not high enough the system might have crashed at $T < T_{gss}$ or the states to be saved might be corrupted which defeats the whole purpose of global state saving.

4.2.2 MT-based evaluation

The *Mission-Time* is defined as the time interval where $R(T) \geq R_{min}$, therefore:

$$R(MT) = R_{min}$$

The values of MT, as obtained from Equation 4.1.13, are plotted in Figures 4.3 and 4.4 as a function of N for different values of c. The same results are reported numerically in Table 4.1.

These curves show that for a given value of c, there exists a value of N at which the MT is maximal. We denote this value by N_p.

$$MT(c, N_p) \geq MT(c, N) \quad \forall N$$

It is clear from these curves that for smaller values of N ($N < N_p$) the inherent redundancy of the system provides a higher mission time. As N increases ($N > N_p$) the higher failure rate dominates and reduces the mission time. Furthermore, as c is increased, the value of N_p also increases (see Figure 4.4).

From the analysis of these results, we can deduce the following:

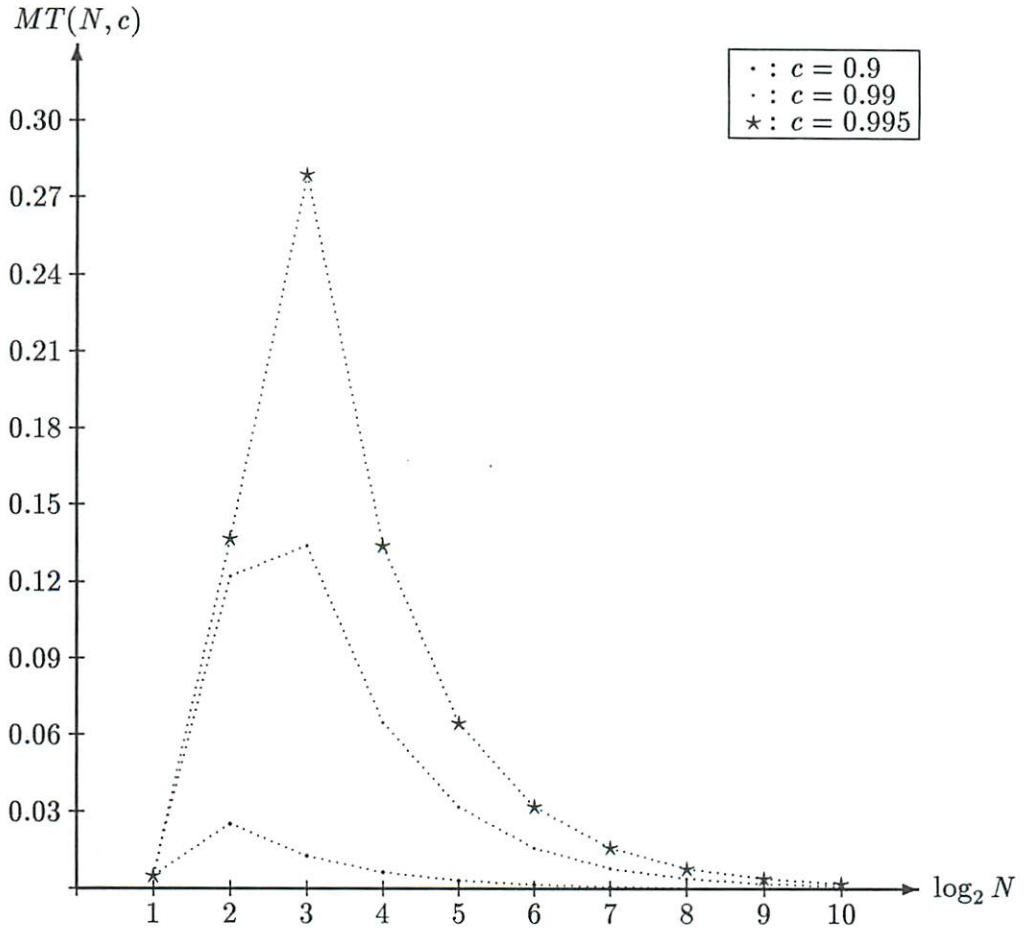


Figure 4.3: Mission-Time as function of N ($D = N - 1$, $c < 0.995$)

N	c			
	0.9999	0.999	0.99	0.9
2	0.11	0.105	0.096	**0.044**
4	0.38	0.37	**0.232**	0.025
8	0.82	0.75	0.134	0.013
16	1.37	**0.93**	0.065	0.0063
32	1.95	0.38	0.032	0.0032
64	**2.51**	0.17	0.016	0.0016
128	1.54	0.08	0.008	0.0008
256	0.5	0.04	0.004	0.0004
512	0.22	0.02	0.002	0.0002
1024	0.11	0.01	0.001	0.0001

Table 4.1: Mission-Time $D = N - 1$ and $R_{min} = 0.99$

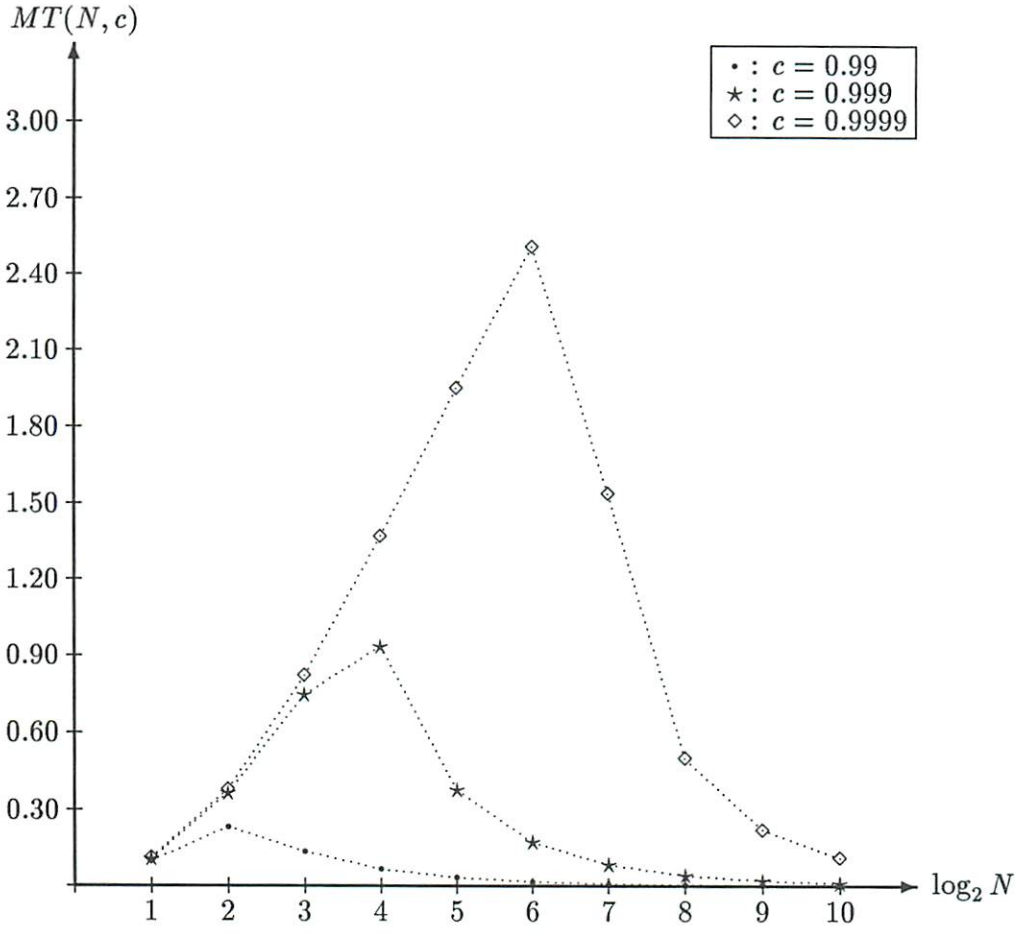


Figure 4.4: Mission-Time as function of N ($D = N - 1$, $c > 0.99$)

- The plot of the mission-time as a function of the number of processors, N, for a gracefully degradable system shows that the mission-time reaches a peak value at $N = N_p$.
- The peak value of the mission-time is significantly larger than that of a single processor. For example: for $R_{min} = 0.99$ the mission -time of a single processor, $MT(1) = 0.01$, therefore $MT(N_p, 0.999) \geq 93MT(1)$.
- As the number of processors is increased beyond N_p, i.e $N > N_p$, we observe a decrease in the mission-time. This decrease can be observed to be inversely proportional to N. That is $MT(2N, c) = 0.5MT(N, c)$.
- While the peak value of the mission-time of a multicomputer system can be significantly larger than that of a single processor, the reverse becomes true for very large values of N. For example, from Table 4.1 we can observe that $MT(1) = 10MT(1024, 0.99)$.
- From Table 4.1 we can observe that, for $N > N_p$, a 10 fold decrease in $(1-c)$ (the probability of failed recovery) results in a 10 fold increase in the MT for the same number of processors. For example, $MT(128, 0.999) = 10MT(128, 0.99)$. In other words, the mission-time is inversely proportional to $(1 - c)$.

From these numerical results we can deduce the following proportionality expression:

$$MT \propto \frac{1}{(1-c)N} \tag{4.2.16}$$

While this expression is empirically based on the numerical results for $D = (N-1)$ and $R_{min} = 0.99$, it will be analytically justified in section 4.3.3.

An MT based evaluation of T_{gss} implies that an increase in N is not always beneficial to the total computation time. In fact, for $N > N_p$ it could be detrimental since the T_{gss} interval is reduced as N increases while the time overhead of GSS procedure itself increases too. However, since $R(T_{gss}) \geq R_{min}$ this approach, offers the advantage of guaranteed high reliability when the GSS procedure is performed and therefore the correctness of the saved states.

This analysis, however, does not indicate whether the decrease in T_{gss}, for $N > N_p$, is offset by the increase in execution speed, and therefore of computational work. This problem is addressed in the next section where we analyze the effects of c on the measure of *computational work* and *reliable computational work.*

4.2.3 Expected Number of Failures

In this section we propose an analytical evaluation of the expected number of failures in the time interval $[0, MT]$. This analysis will allow us to derive the value of N_p where the MT is maximal as observed in section 4.3.2.

Because of the cumulative effects of the probability of successive recovery, after the i^{th} failure, the reliability of the system is constrained by:

$$R(t) \leq c^i$$

Let K' be defined such that:

$$c^{K'} = R_{min}$$

Therefore, an integer value of K', K, can be derived as:

$$K = \left\lfloor \frac{\log R_{min}}{\log c} \right\rfloor \tag{4.2.17}$$

Given the definition of MT, K' is the expected number of failures in the interval $[0, MT]$ constrained by the condition that $D \geq K'$. In other words, if the number of processors is large enough, K failures are sufficient to reach $R(t) = R_{min}$. Since K, as defined in equation 4.2.17, can take only integer values, it is an approximation of the expected number of failures K'.

From this analysis, we can deduce that when the number of allowed degradation states is sufficiently large (i.e, $D \gg K$), the necessary condition to reach the minimum reliability level ($R(t) = R_{min}$) and therefore the mission-time is that K processors fail. Since the rate of failures is proportional to the number of processors, the time interval $[0, MT]$ is inversely proportional to N. On the other hand, for ϵ very small and $x = 1 - \epsilon$, we can use the following approximation for $\log x$:

$$\log x \approx 1 - x$$

We have therefore demonstrated that for $D \gg K$:

$$MT \propto \frac{1}{(1-c)N}$$

4.3 Computation-Based Analysis

In this section we present an evaluation of performance and reliability of large-scale degradable systems based on the notion of *computational work*. There is no formally defined unit of computational work. In this analysis we will use *processor-hours* as units of computational work. Other measuring units could be machine instructions. Any computational task is characterized by a certain amount of computational work, measured in processor-hours. When this task is executed over several processors, the execution time is reduced, but the amount of processor-hours required for that computation is kept constant if the speed-up is linear. For non-linear speed-ups the amount of required processor-hours increases due to added overhead.

Let T_n be the execution time of a given computation over n processors. S_n is the attainable speed-up defined by:

$$S_n = \frac{T_1}{T_n}$$

S_n is equivalent to the number of *effective processors* (i.e the number of virtual processors fully utilized by the given computation). We define $CW(N,t)$ as the amount of *effective computational work* a system will deliver for a given computational speed-up.

$$CW(N,t) = \int_{\tau=0}^{t} \sum_{i=0}^{D-1} S_{(N-i)} P_i(\tau) d\tau \tag{4.3.18}$$

Based on the model described in section 4.2, we define $PH(N,t)$ as the amount of processor-hours a system, with initially N processors, can deliver up to time t, as:

$$PH(N,t) = \int_{\tau=0}^{t} \sum_{i=0}^{D-1} E_{(N-i)}(N-i) P_i(\tau) d\tau \tag{4.3.19}$$

For a computation that exhibits linear speed-up (i.e. $S_n = n$) we have: $CW(N,t) = PH(N,t)$. For the sake of simplicity, in the rest of this discussion, we will assume a best case of linear

speed-up and therefore $E_N = 1$. Note that $PH(N,\infty)$ is the *mean computation before failure* (MCBF) and $CW(N,t)$ is the integral of the *computational availability*, $a_C(t)$, as defined in [Bea78]. Both $PH(N,t)$ and $CW(N,t)$ are *expected* values of the processor-hours and computational work measures.

In section 4.4.1 we show that, in a gracefully degradable system, the amount of computational work, $PH(N,t)$, is upper-bounded. The upper-bound is independent of N, i.e

$$PH(N,t) < PH_{max} \qquad \forall N, \forall t$$

Using $PH(N,t)$, we define, in section 4.4.4, the measure of *Reliable Computational Work*, $RCW(N)$, and show how it can be used to evaluate T_{gss}.

4.3.1 Upper Bound on PH

In this section, we prove that the amount of computational work a purely degradable system can deliver is upper bounded and that the upper bound is independent of the initial number of processors. We derive this upper bound using (1) a discrete analysis and (2) a continuous-time Markov model.

Theorem 4.1 *$\forall N$ and $c < 1$ $\exists$ PH_{max} such that*
$PH(N,t) < PH_{max}$, $\forall t$.

4.3.1.1 Proof 1: Continuous-time Markov model

Using the binomial theorem, Equation 4.1.11 can be rewritten as:

$$P_i(t) = c^i \binom{N}{i} \sum_{k=0}^{i} \binom{i}{k} (-1)^{(i-k)} (e^{-\lambda t})^{(N-k)} \tag{4.3.20}$$

Therefore, Equation 4.3.19 can be rewritten as:

$$PH(N,t) = \int_0^t \sum_{i=0}^{D-1} (N-i) c^i \binom{N}{i} \sum_{k=0}^{i} \binom{i}{k} (-1)^{(i-k)} (e^{-\lambda \tau})^{(N-k)} d\tau \tag{4.3.21}$$

Integrating over τ, and taking the limit as $t \longrightarrow \infty$, we obtain:

$$PH(N,\infty) = \frac{1}{\lambda} \sum_{i=0}^{D-1} (N-i) c^i \binom{N}{i} \sum_{k=0}^{i} \binom{i}{k} (-1)^{(i-k)} \frac{1}{(N-k)} \tag{4.3.22}$$

The second summation in Equation 4.3.22 can be transformed using binomial identities into:

$$\sum_{k=0}^{i} \binom{i}{i-k} (-1)^{(i-k)} \frac{1}{(N+i)+(i-k)} = \frac{1}{(N-i)\binom{N}{i}}$$

Equation 4.3.22 is reduced to:

$$PH(N,\infty) = \frac{1}{\lambda} \sum_{i=0}^{D-1} c^i = \frac{1-c^D}{\lambda(1-c)} \tag{4.3.23}$$

Therefore:

$$PH_{max} = \frac{1}{\lambda} \frac{1}{1-c} \tag{4.3.24}$$

and $PH(N,\infty) < PH_{max} \quad \forall N$. □

4.3.1.2 Proof 2: Discrete Analysis

The amount of processor-hours, PH, can also be expressed as a function of the number of failures i. $PH(i)$ is therefore the amount of processor-hours at the i^{th} failure.

$$PH(1) = \frac{N}{N\lambda} = \frac{1}{\lambda}$$

$$PH(i) = cPH(i-1) + \frac{N-i-1}{(N-i-1)\lambda} = cPH(i-1) + \frac{1}{\lambda} \tag{4.3.25}$$

From Equation 4.3.25 we can rewrite $PH(i)$ as:

$$PH(i) = \frac{1}{\lambda}(1 + c + c^2 + ... + c^{(i-1)}) = \frac{1}{\lambda}\frac{1-c^i}{1-c} \tag{4.3.26}$$

Therefore:

$$PH_{max} = \lim_{i \to \infty} PH(i) = \frac{1}{\lambda}\frac{1}{(1-c)} \tag{4.3.27}$$

Therefore PH_{max} is a constant upper bound on $PH(i)$ as $i \longrightarrow \infty$. □

The conclusion from Theorem 1 is that no matter how large the initial number of processors is, there is an upper bound on the amount of processor-hours that are obtainable when $c < 1$. This upper bound is determined by c only and is reached asymptotically.

PH_{max} is therefore the upper limit on the *mean computation before failure* (MCBF). Comparing this result to that in section 4.3.1 shows that while the expected time to failure of a degradable system increases logarithmically with N, the expected computational work performed in that interval is upper bounded for all N. Therefor increasing the system size does not increase the amount of expected computational work the system can deliver before total failure.

4.3.2 Reliable Processor-Hours

The measure of *reliable processor-hours*, RPH, is defined as the amount of processor-hours available while the reliability is maintained above a given minimum, i.e:

$$RPH(N,c) = PH(N, MT) = \int_{t=0}^{MT} \sum_{i=0}^{D-1} (N-i)P_i(t)dt \tag{4.3.28}$$

The results of evaluating the RPH, according to Equation 4.3.28, are presented in Table 4.2, for $D = N/2$, and Table 4.3 for $D = N-1$. The values of c have been chosen in the range $[1, 0.99]$. The value of R_{min} has been set to 0.99. The values of $RPH(N,c)$ are expressed in $processor - hours$ where the unit time is taken as $1/\lambda$.

Two observations can be made:

1. for a given value of c, there is a value of N beyond which an increase in N will not increase the amount of reliable processor-hours. We denote this value by $N_{ph}(c)$. For example, $N_{ph}(0.999) = 64$. We can formally define $N_{ph}(c)$ as:

$$\forall c < 1, \quad N \geq N_{ph}(c) \Longrightarrow RPH(N,c) = RPH_{max}(c)$$

2. the values of N_{ph} and RPH_{max} increase with increasing values of c. For example, in Table 1, $N_{ph}(0.99) = 16$ and $RPH_{max} = 1.0$, $N_{ph}(0.999) = 64$ and $RPH_{max} = 10.0$ and $N_{ph}(0.9999) = 512$ with $RPH_{max} = 100$.

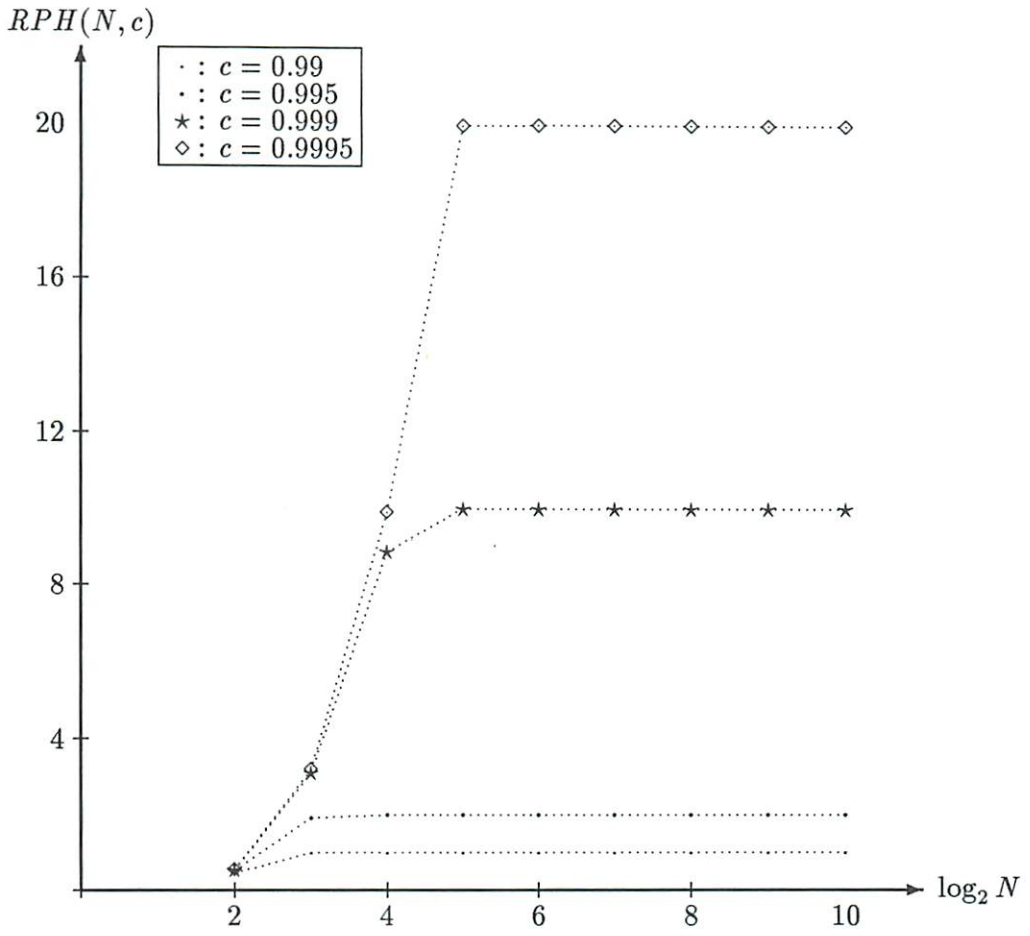


Figure 4.5: RPH as function of N and c

N	c					
	1.0	0.9999	0.9995	0.999	0.995	0.99
4	0.168	0.168	0.168	0.168	0.164	0.156
8	0.966	0.963	0.955	0.94	0.84	0.70
16	3.38	3.35	3.29	3.2	1.95	**1.00**
32	9.2	9.15	8.8	8.0	**2.00**	1.00
64	22.4	22.1	19.0	**10.0**	2.00	1.00
128	50.3	49.0	**20.0**	10.0	2.00	1.00
256	106.7	98.0	20.0	10.0	2.00	1.00
512	220	**100**	20.0	10.0	2.00	1.00
1024	455	100	20.0	10.0	2.00	1.00

Table 4.2: Reliable Processor-hours for $D = N/2$ and $R_{min} = 0.99$

N	c					
	1.0	0.9999	0.9995	0.999	0.995	0.99
4	0.563	0.562	0.558	0.553	0.511	0.459
8	3.28	3.26	3.19	3.1	1.93	**1.0**
16	10.42	10.33	9.89	8.86	**2.0**	1.0
32	25.93	25.62	19.98	**10.0**	2.0	1.0
64	57.67	56.64	**20.0**	10.0	2.0	1.0
128	122.77	**100.3**	20.0	10.0	2.0	1.0
256	250.7	100.0	20.0	10.0	2.0	1.0
512	508.0	100.0	20.0	10.0	2.0	1.0
1024	1019.0	100.0	20.0	10.0	2.0	1.0

Table 4.3: Reliable Processor-hours for $D = (N-1)$ and $R_{min} = 0.99$

	c				
	0.9999	0.9995	0.999	0.995	0.99
K	100	20	10	5	1
RPH'	99.5	19.9	9.95	1.995	1
RPH_{max}	100	20	10	2	1
Ph_{max}	10^4	2000	1000	200	100

Table 4.4: Comparison of RPH' and RPH_{max} for $R_{min} = 0.99$

Theorem 1 states that the expected amount of processor-hours in the interval $[0, \infty]$ is upper bounded by PH_{max}. These results show that the expected amount of processor-hours in the interval $[0, MT]$ is also upper bounded, the upper bound being RPH_{max}.

RPH_{max} is therefore the maximum expected amount of computational work the system can deliver subject to the constraint of $R(t) \geq R_{min}$. In the next section we present an analytical derivation of $RPH_{max}(c)$.

The maximum value of $RPH(N, c)$, $RPH_{max}(c)$, can be derived analytically by using the expression for the expected number of failures in the interval $[0, MT]$ as defined in section 4.3.3.

$$RPH_{max} \approx RPH' = \frac{1 - c^K}{1 - c} \tag{4.3.29}$$

Note that Equation 4.3.29 is an approximation of RPH_{max} because K can take only integer values while MT in Equation 4.3.28 has real values.

Table 4.3.2 shows the values of K, RPH', RPH_{max} and Ph_{max} for various values of c. From the values of RPH_{max} and RPH', we can observe that the approximation of Equation 4.3.29 is accurate within 5%. Therefore, Equations 4.3.29 and 4.2.17 provide a simple method for deriving RPH_{max}, given the values of c and R_{min}.

4.3.2.1 Discussion of Results

The results of section 4.2 show that there is no increase in reliable computational work when N is increased above N_{ph} for a given value of c. This confirms the results obtained in the MT based evaluation. Although the data reported in Figures 4.3 and 4.4 and in Tables 4.2 and 4.3 covers

only a few values of N, it can be noted that the values of N_p, where MT is maximal, and those of N_{ph}, where $RPH(N, c)$ is constant, are very closely related.

It appears, therefore, that for a given value of the coverage c there exists an optimal value of N, N_{opt}, that would maximize the mission-time MT, and therefore the interval T_{gss}, while preserving a high reliability. From the numerical data presented in this chapter we cannot derive an *exact* value for N_{opt}, however, the interval can easily be narrowed down. For example, for $c = 0.999$ and $D = N - 1$ we have:

$$16 \leq N_{opt} < 32$$

4.3.3 Reliable Computational Work

RPH evaluates the amount of reliable processor-hours potentially available from the system. The fraction of RPH that is actually used by a computation depends on the speed-up S_n of the computation. The speed-up of a computation is equivalent to the number of *effective processors* or the number of *virtual fully-utilized* processors. It is defined as:

$$S_n = \frac{T_1}{T_n}$$

where T_i is the execution time over i processors. When $S_n = n$ the computation is said to exhibit linear speed-up. This implies that the communication and synchronization overhead in that computation is negligible compared to the execution time. When $S_n < n$ the speed-up is said to be sub-linear.

Similarly to RPH we define RCW as:

$$RCW(N, c) = CW(N, MT)$$

RCW is therefore the amount of *effective reliable computational work* a system can deliver with respect to a given computation while $R(t) \geq R_{min}$. Therefore $RPH = RCW$ for $S_n = n$.

In evaluating RCW, we will take as example a sub-linear speed-up case where:

$$S_n = \frac{n}{\log n}$$

The results, plotted in Figure 4.6, show that:

$$\exists N_r \ \ such\ that \ \ RCW(N_r) \geq RCW(N) \quad \forall N$$

In other words, there exists a value of N denoted by N_r at which the value of RCW is maximal.

Figure 4.7 shows the plot of both RPH and RCW versus N for $c = 0.995$ and $S_n = \frac{n}{\log n}$. It can be noted that $N_r = N_{ph}$ as reported in Table 4.3 (for $D = N - 1$). This effect is predictable: since RPH is constant for $N > N_{ph}$ and a linear speed-up, for a sub-linear speed-up RCW must be a decreasing function of N.

This implies that as the system size is increased over N_{ph}, the probability of a computation not completing reliably decreases if the speed-up of the computation is sub-linear.

This result has implications on the scalability of graceful degradation. For a large-scale gracefully degradable system to be scalable, any increase in the system size should be matched by an increase in the quality of the recovery scheme, i.e the coverage factor, in order to maintain the same performability level.

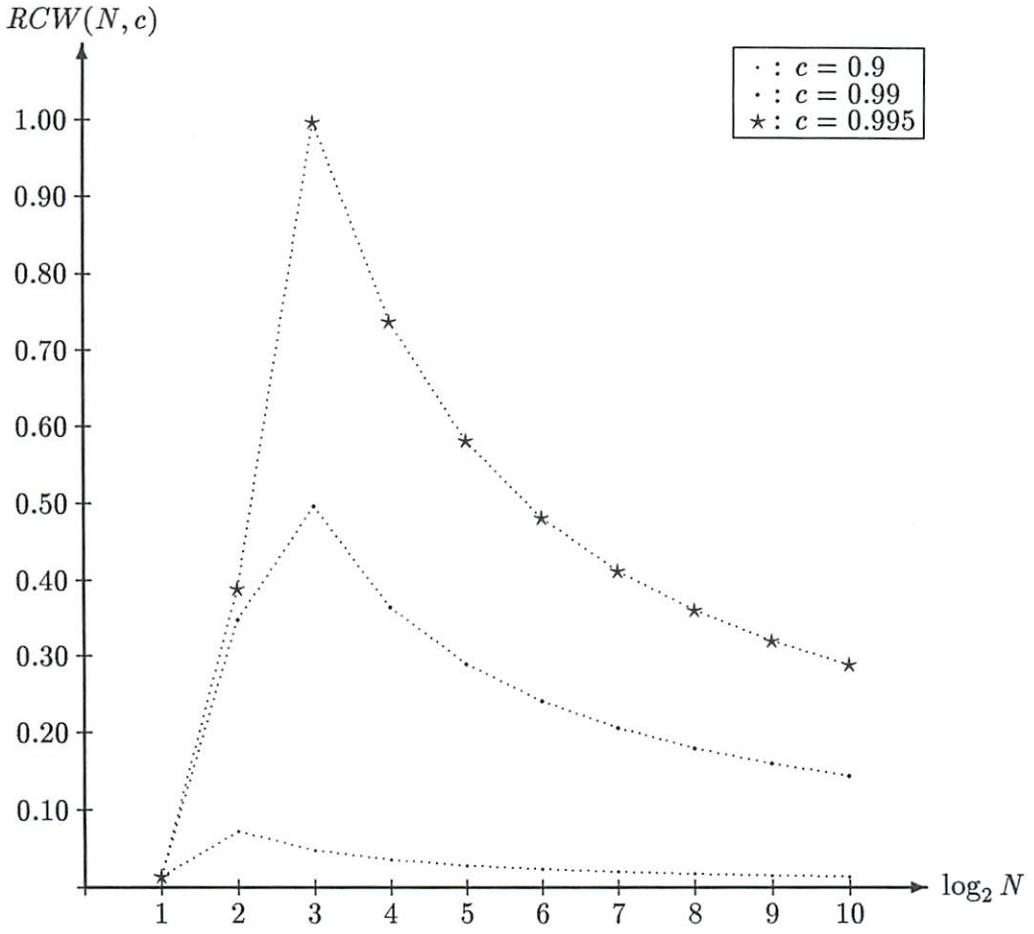


Figure 4.6: RCW as function of N and c for $S_n = \frac{n}{\log n}$

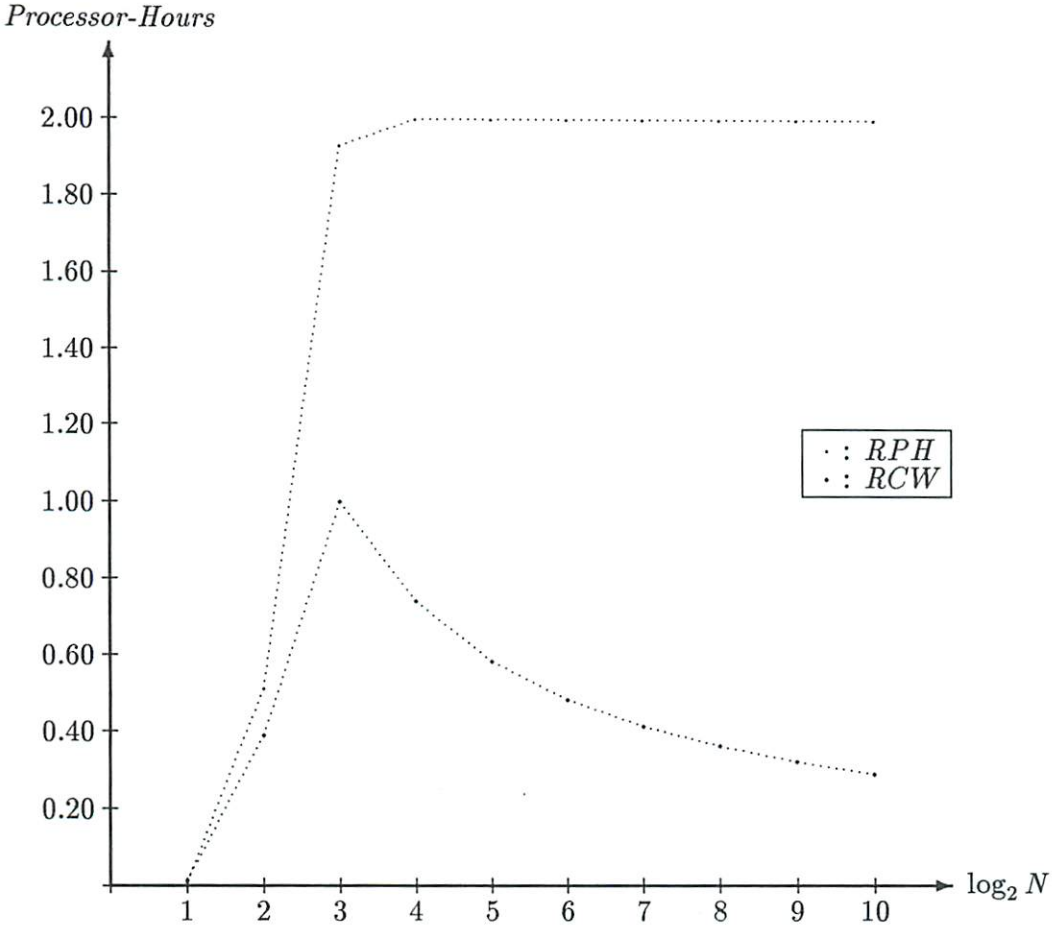


Figure 4.7: RCW and RPH as function of N, $c = 0.995$ and $S_n = \frac{n}{\log n}$

4.4 Discussion of the Results

The results of the analysis performed in this chapter point out to the fact that gracefully degradable large-scale systems do not scale-up. If a minimum reliability level is to be maintained throughout, then there exists a system size that would provide the optimal mission time and amount of reliable computational work. For any larger system there would be a decrease in either performance (as expressed in computational work) or reliability (as represented by the minimum level).

Throughout this analysis, one parameter has been maintained constant and used a measure of unit time, that is the mean-time-to-failure of the single processor ($MTTF_1$). We have therefore assumed that the systems were built using the same set of basic building blocks. A system architect, however, has a wide range of choices for his basic building blocks. The technologies range from high-speed high-power ECL technology to low-speed low-power CMOS technologies. In addition to the speed and power consumption factors, technologies can also be classified by their age. Old technologies have time-proven characteristics and well established design rules and methodologies. Their parameters have typically very narrow tolerance levels and they generally have a very low failure rate. New technologies, on the other hand, are more prone to either design errors or higher component failure rates.

Therefore, the choice of a technology by a system designer not only determines the allowable switching speed, and thereby the potential computing power of the system, but also the expected rate of failure [1]. By determining the failure rate, the age of a technology determines the assumed unit-time $MTTF_1$. While new technologies can offer switching speeds several orders of magnitude larger than the older technologies, these can often be several orders of magnitude more reliable. It is conceivable, therefore, that a system built with a time-proven but slow-switching technology might actually outperform a system built with faster but less reliable technology.

[1] Another major relevant factor to this choice is the cost of design and components associated with a given technology. This factor, however, is not of immediate relevance to the analysis proposed in this chapter.

Chapter 5

ALGORITHM FOR DISTRIBUTED FAULT-TOLERANCE

"We have another bad ... unit. My fault predictor indicates failure within twenty-four hours".

"I don't understand it, Hal. Two *units can't blow in a couple of days."*

"It does seem strange, Dave. But I assure you there is an impending failure."

Arthur C. Clarke
2001: A Space Odyssey

In Chapter 4, we have shown that the performance and reliability of gracefully degradable systems do not scale up as the number of processing elements is increased, unless the quality of the protection scheme, i.e. the coverage factor, is increased accordingly. A highly fault-tolerant protection scheme for large-scale systems should be immune to the effects of node disconnections. In fact, the results in Chapter 3 show that the probability of disconnection can become a significant threat in large systems with constant connectivity.

In this chapter we propose a methodology for implementing distributed checkpointing and an algorithm based on that methodology, that implements distributed fault-tolerance. The methodology for distributed checkpointing is based on the semantic properties of functional program execution.

We review the concepts and properties of functional program execution and present the methodology that allows program checkpointing to be implemented in a distributed fashion at run-time. A simple iterative distributed algorithm for fault-tolerance based on functional program execution is then described. The performance of this algorithm is then evaluated and techniques for implementing system level diagnosis, based on this algorithm, are proposed.

5.1 Distributed Checkpointing in Functional Execution

In this section we describe a methodology that allows the distributed checkpointing of programs at run-time. This methodology is based on the Church-Rosser property of functional program execution. In section 5.1.1 we review some of the basic concepts and properties of functional languages and functional program execution. The proposed methodology is described in section 5.1.2.

5.1.1 Functional languages and execution

Functional languages have been proposed as an alternative to imperative languages mainly because of their ability to exactly model mathematical expressions [Bac78]. A functional language consists of a set of primitive functions and primitive data objects (atoms). A set of operators allow operations on functions such as composition, inversion, and the construction of complex data objects such as lists. A program in a functional language consists of a set of function applications on some data objects. Unlike imperative languages, the execution of functional languages is *side-effect free.*

The various properties of functional languages have been widely discussed in the literature. These include the possibility of algebraic program verification and program derivation from specifications. Probably the most popular and most interesting property as far as program execution is concerned is the Church-Rosser property which is an immediate consequence of the property of side-effect free execution. This property states that the order of function application in the execution of a functional program does not have any effect on the outcome of the program.

While the Church-Rosser property was derived in the context of functional languages, it is not a characteristic of the language itself as much as a that of the *execution model.* In fact, this property holds as long as side-effect free execution is guaranteed. In a *functional execution model*, a program is modeled as a set of side-effect free tasks. A task executes on a set of input data objects and produces a set of output data objects in a function like way. This property, therefore, holds irrespective of the actual programming language used as long as the functionality of the execution is preserved.

Based on the referential transparency property, we propose a model for the functional execution of distributed programs on multicomputer systems. In this model a program executes as a set of distributed communicating tasks. The task is the smallest execution entity; it executes as a process on a single processor. Data is communicated between tasks as messages. All incoming messages are received by a task before execution starts. All outgoing messages are sent upon completion of execution. Task execution is functional in that it has no effect on the program except through its outgoing messages. In the next section, we describe how such a model can support distributed checkpointing.

5.1.2 A Distributed Checkpointing Methodology

A reliable checkpointing mechanism is an essential element in any fault recovery scheme. In a uniprocessor, a consistent checkpoint is a snapshot of the state of the program at a given time. It can be achieved, conceptually, by saving the state of the memory and all relevant registers. In a distributed system, however, the problem of checkpointing is rendered more complex by the asynchronous nature of the execution and the communication delays among processors.

The methodology for distributed checkpointing presented in this section was originally proposed as the Token Resending scheme by Gaudiot *et al.* [GR85] for the reliable execution of data-flow programs.

The scheme is as follows: whenever a task completes execution, a copy of all the output messages that are produced is kept by the processor on which the task executes. At the same time, a special acknowledgement message (*ack*) is sent to all the processors from which messages have been received as input messages to that task. The reception of an *ack* message results in the deletion of the corresponding copy of the message. Figure 5.1 illustrates this scheme. Task C starts executing on processor P_3 when both tasks A and B have terminated and messages containing the data values x and y are received. Copies of the values of x and y are kept as x' and y' in processors P_1 and

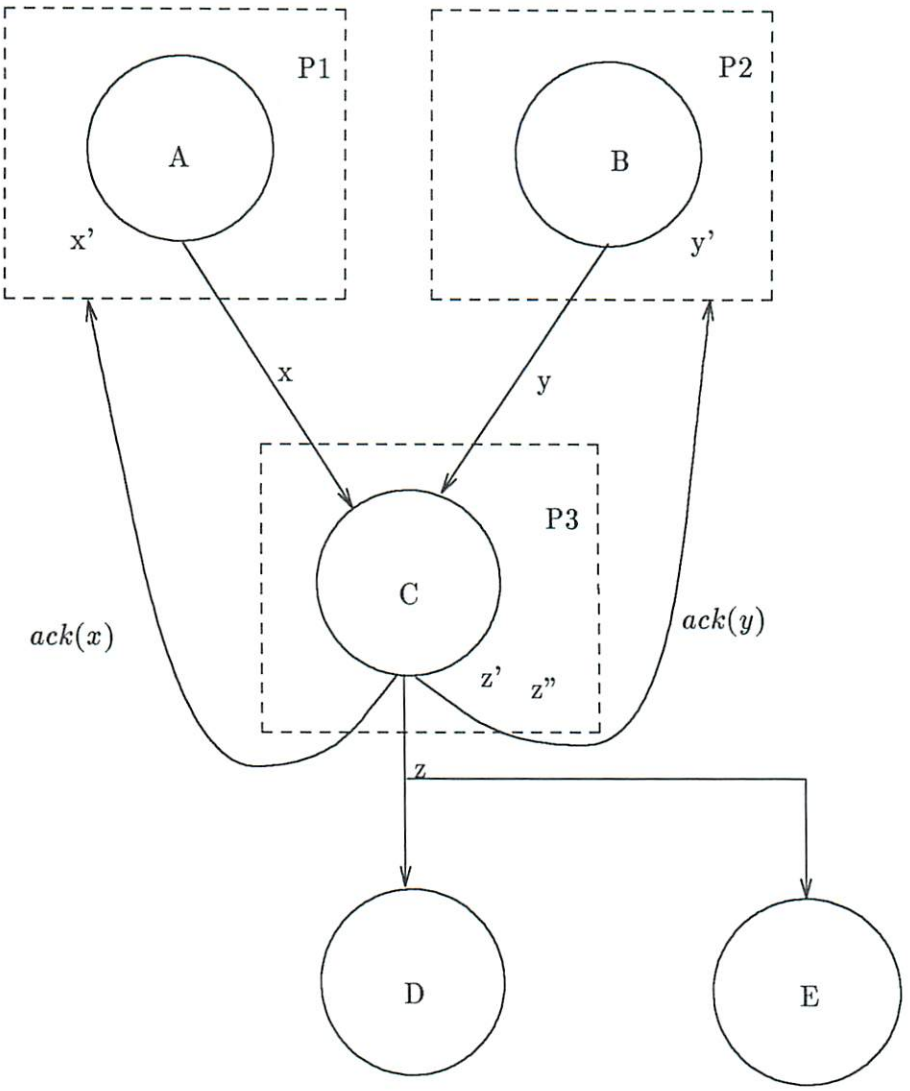


Figure 5.1: Example of distributed checkpointing

P_2 respectively. When task C terminates, the result value z is sent to tasks D and E. Two *ack* messages are sent to processors P_1 and P_2. That results in the deletion of the backup values x' and y'. If processor P_3 had failed before the completion of task C and the failure was detected and notified to processors P_1 and P_2, then task C could have been restarted on another processor with the same set of input data. Note that since the data value z is sent to two different destinations, two backup copies, z' and z'' are kept in P_3. A constraint imposed by this mechanism on task allocation is that two successive tasks should not be allocated to the same processor. Whenever, for efficiency considerations, two successive tasks must be allocated to a same processor, then they are are checkpointed as a single combined task.

Since any task that has not completed execution successfully can be restarted at any time from the backup copies of its input data set, the set of all backup data values constitutes the global state, or checkpoint, of the program. This checkpoint can be visualized as a wavefront advancing through the program execution graph. This mechanism allows for the checkpointing to be performed asynchronously at run-time along with the program execution. It does not, however, allow for the detection of failures in the system. The algorithm proposed in the next section exploits the properties of functional execution to implement fault detection and recovery.

5.2 An Iterative Algorithm

In this section we propose a distributed algorithm that is based on the checkpointing methodology described in section 5.1.2. This algorithm relies on both dual space redundancy and multiple time redundancy. the redundant execution serves the dual purpose of (1) improving the reliability of a task execution, and (2) detecting failures or transient faults.

Figure 5.2 is a description of the algorithm. Every task on pairs of processors and the results are compared. If a match occurs, the execution of the program proceeds by sending the output data to successor tasks. If the results do not match, the task is re-executed on another pair of processors. At every iteration where no match occurs, an *iteration-number* is incremented and passed along with the input data set. This number is used to determine the address of the next pair of processors in case of a failed match. In this fashion, a task is not executed twice on the same pair of processors. The *error-number* value is also incremented at each mismatch and is used to implement diagnosis algorithms that are described in section 4.

This algorithm has the following properties:

- Being an iterative algorithm, the outcome at each iteration is independent of the previous ones, therefore there is no reliance on a recursive history or results to determine a possible match as in the RAFT algorithm [Agr85].
- By relying on both time and space redundancy, the algorithm allows the detection of both transient and permanent faults.
- The algorithm has a relatively high cost of hardware overhead. The number of processors used after iteration i is $2i$ whereas it is $i+1$ in the recursive algorithm. However, this cost is still substantially smaller than that of a TMR scheme where a voter device would be needed.

The fact that the iterative algorithm does not rely on a recursive data structure of previous outcomes makes it suitable for distributed execution. Multicomputer systems are often structured as regular topologies such as a mesh, binary n-cube or cube-connected-cycles. The pairing of

processors in such topologies is quite simple, a distributed scheme for the allocation of successive iterations is described in section 5.4.

In the occurrence of a hard failure, a processor might not output any result. This situation can easily be detected by the other processor in the pair by using a time-out mechanism. A well designed time-out interval would take into account any discrepancy in the execution time between the two processors or possible network delays.

5.3 Performance Evaluation

In this section, we derive the probabilities of the various outcomes at each iteration as well as the expected number of iterations. We will show that under normal conditions, these results compare favorably with those reported by Agrawal for the recursive algorithm [Agr85].

At each iteration of the algorithm, the possible outcomes are:

- *no-match*
- *match*, in which case two conditions are possible:
  - *correct result*
  - *incorrect result*

The following quantities are defined for a given task executing on a given processor. Let p be the probability that a task executes correctly; n the number of possible failure modes and q_i the probability of a failure in mode i, where $i = 1 \ldots n$. Therefore:

$$1 - p = \sum_{i=1}^{n} q_i$$

Let p_m be the probability of a match at a given iteration. Note that since the outcome at any iteration is independent of the outcomes at the previous iterations, p_m is independent of the number of iterations. We can write:

$$p_m = p^2 + \sum_{i=1}^{n} {q_i}^2 \tag{5.3.30}$$

A match can occur when both processors have the correct result or the same failure mode with respect to that computation. For simplicity we assume that $q_i = q \quad \forall i = 1 \ldots n$, therefore:

$$p_m = p^2 + \frac{(1-p)^2}{n} \tag{5.3.31}$$

Let $P_m(I)$ denote the probability of a match at the I^{th} iteration and I_{exp} the expected number of iterations, then:

$$P_m(I) = p_m(1 - p_m)^{(I-1)}$$

$$I_{exp} = \sum_{I=1}^{\infty} I P_m(I) = \sum_{I=1}^{\infty} I p_m(1 - p_m)^{(I-1)} = \frac{1}{p_m}$$

The quality of the decision made at any iteration is denoted by QD which is the conditional probability of a correct match given that a match occurred, therefore:

$$QD = \frac{p^2}{p_m} = \frac{p^2}{p^2 + \frac{(1-p)^2}{n}}$$

The improvement in quality brought by the algorithm over a non-redundant execution is evaluated by the Quality of Decision Improvement Factor ($QDIF$) which is defined as:

$$QDIF = \frac{1-p}{1-QD}$$

Where $(1-p)$ is the probability of an incorrect result on one processor and $(1-QD)$ is the probability of a match with incorrect results. Replacing with the expression for QD we obtain:

$$QDIF = n\frac{p^2}{1-p} + (1-p)$$

for large p ($p > 0.5$) $QDIF$ can be approximated by:

$$QDIF \approx n\frac{p^2}{1-p}$$

Figure 5.3 shows the plots of I_{exp} against p for different values of n. It can be observed that for $p \geq 0.3$ the value of n practically no effects on I_{exp}. This implies that I_{exp} has a low sensitivity to n and therefore the performance of the system is essentially independent of the number of possible failure modes.

Figure 5.4 shows the plots of QD against p for various n. As would be expected, the quality of the decision is superior for $n = 1000$. This is due to the fact that the number of failure modes increasing decreases the probability of an incorrect match. The same effect can be noticed in Figure 5.5 where $QDIF$ is plotted as a function of n and p. This figure shows $QDIF$ to be an exponentially increasing function of p.

Figure 5.6 shows a comparison of the expected number of trials in the iterative and recursive algorithms for $n = 100$. It demonstrates that for $p < 0.5$ the recursive algorithm is superior to the iterative one. However, under normal conditions, it can safely be assumed that $p > 0.5$, in which case the two algorithms have comparable performances. Since, in the recursive algorithm, the result at each iteration is checked against *all* the previous outcomes, the probability of a *match on incorrect result* increases with the number of iterations. Therefore, for low values of p, the expected number of iterations in the recursive algorithm is low because of the increased probability of a match on incorrect results.

In the iterative algorithm, the probability of a match is independent of the number of iterations which results in a large I_{exp} for low values of p. This characteristic has the effect of *reducing* the probability of an incorrect match when p is low given a maximum number of iterations possible. The maximum number of iterations possible is determined by the number of available processor pairs in the system. The iterative algorithm has the advantage of being less costly in execution (no reliance on a history of previous outcomes) and more suitable for a distributed environment. Note that the expression for QD is independent of the number of iterations unlike the recursive algorithm where the probability of a match at each trial depends on the number of preceding trials.

In Figure 5.7, we compare the quality of the decision in the pair-wise iterative algorithm to that in a similar TMR scheme. The envisioned TMR scheme is similar to the proposed dual redundancy scheme. It consists in executing each task on three processors, instead of two, and producing a result which would be the majority of the three outcomes.

- execute the task on two processors;
- increment *job-counter* in both processors;
- compare the two sets of results;
- if *match* then
  - send results to next tasks;
  - save copy of results as *checkpoint data*;
  - send *ack* messages to the processors where preceding tasks have executed;

  $\Longrightarrow$ preceding *checkpoint data* is deleted;
- else
  - increment *iteration-number* for the task;
  - increment *error-counter* in both processors;
  - send *negative ack* to preceding processors;

  $\Longrightarrow$ task is retried on a different pair of processors;

Figure 5.2: Iterative Distributed Algorithm

	QD, $n = 100$	
p	*Iterative*	*TMR*
0.5	0.9901	0.7952
0.6	0.9956	0.9064
0.7	0.9982	0.9645
0.8	0.9994	0.9901
0.85	0.9997	0.9958
0.9	0.99987	0.9987
0.95	0.99997	0.99980

Table 5.1: QD in the iterative and TMR schemes ($n = 100$).

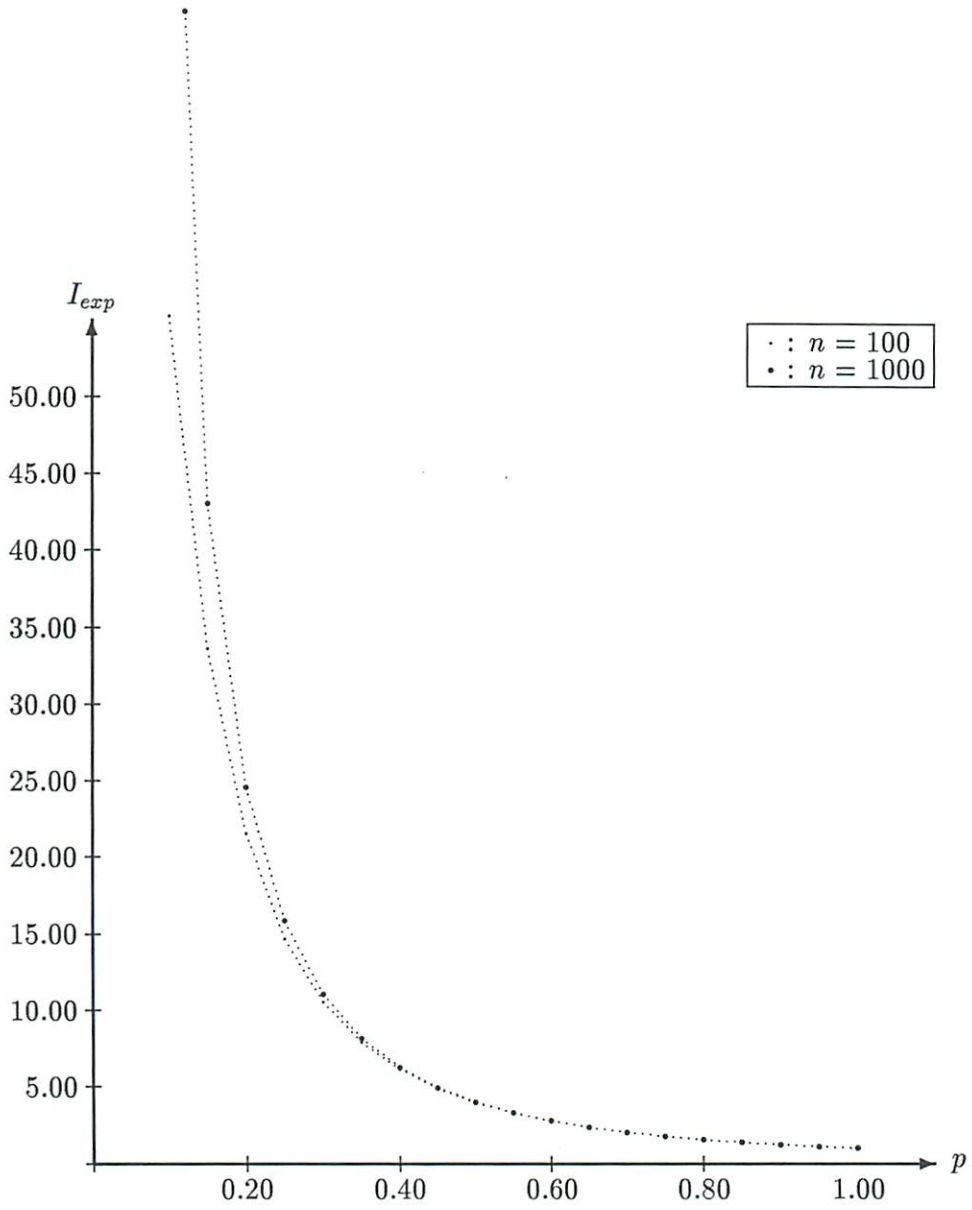


Figure 5.3: Expected number of iterations I_{exp} as function of n and p.

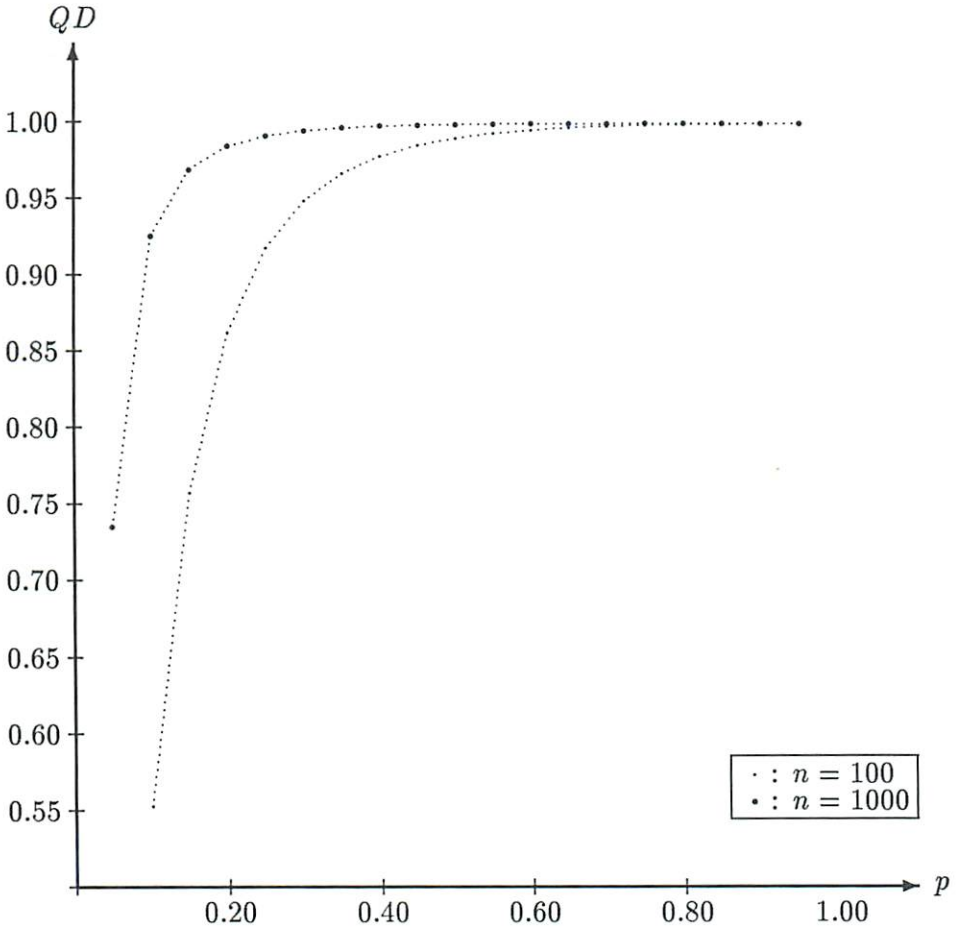


Figure 5.4: Quality of the decision QD as function of n and p

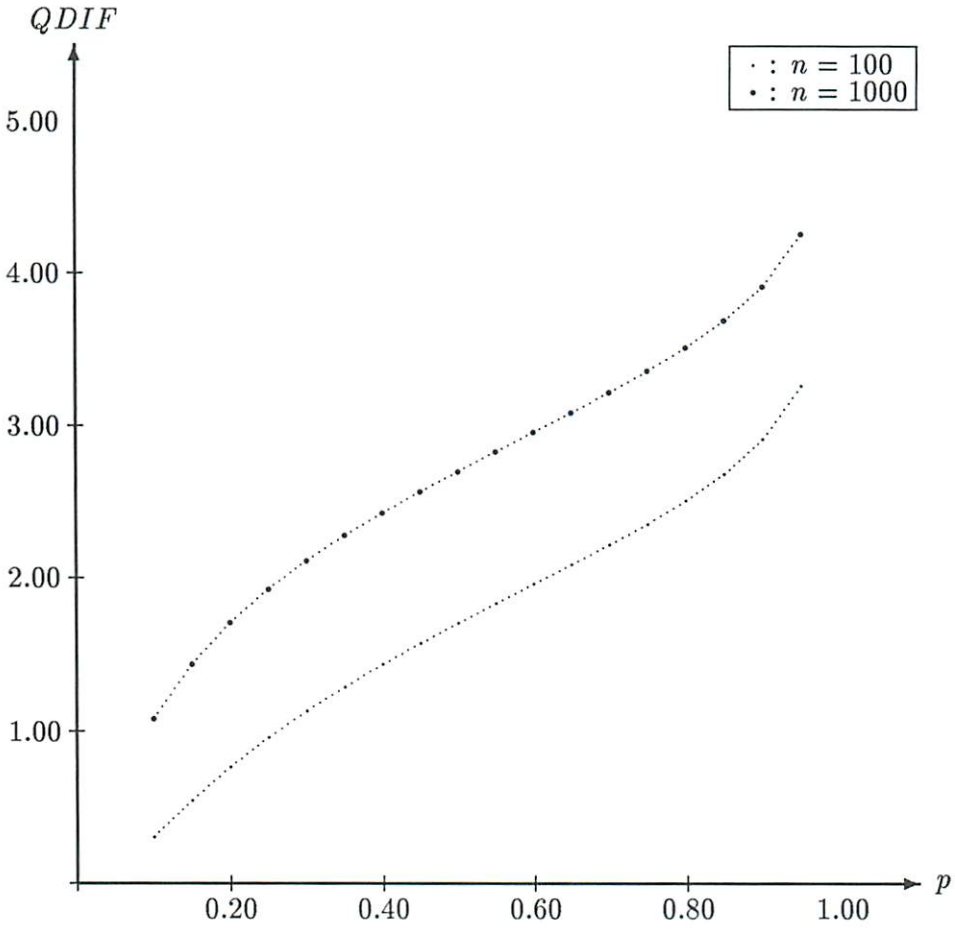


Figure 5.5: $QDIF$ as function of n and p

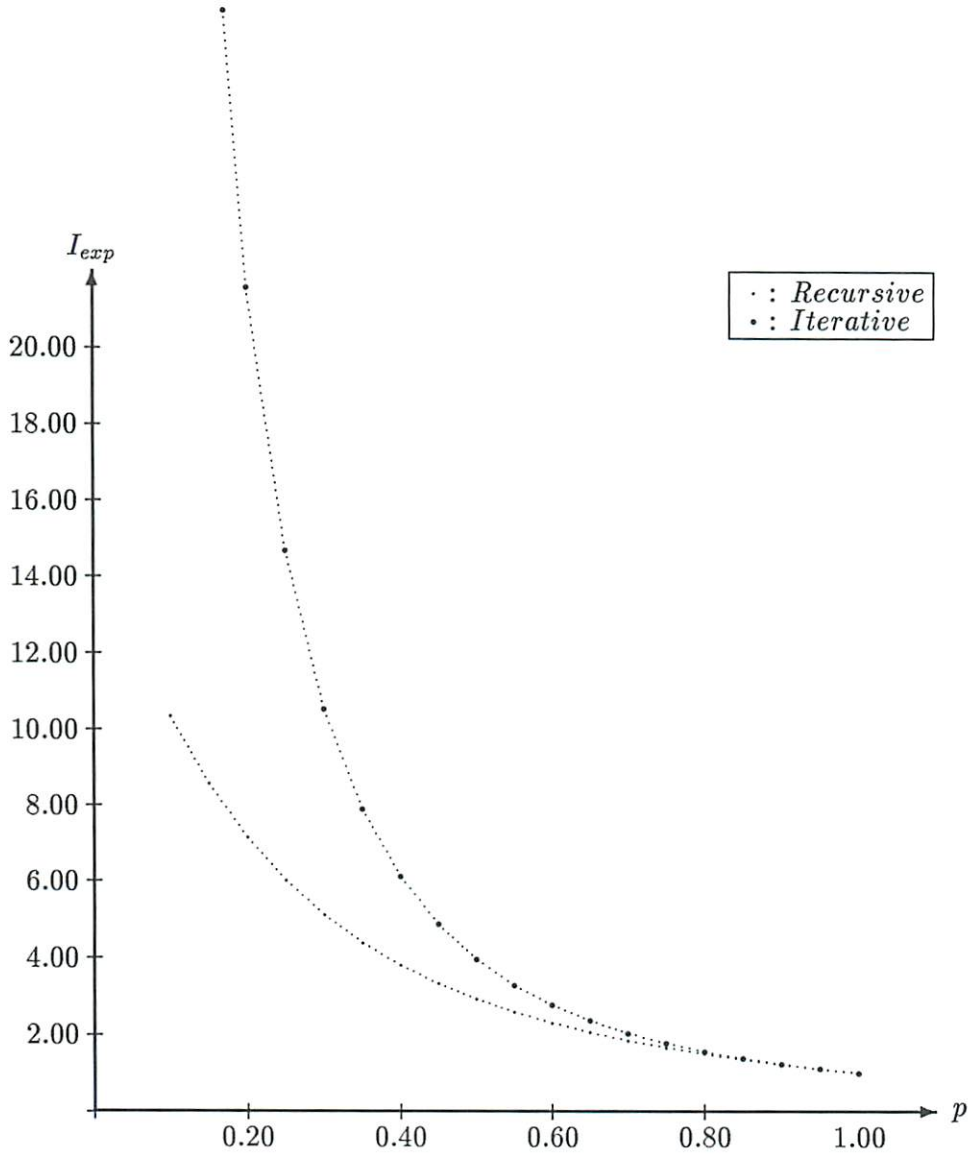


Figure 5.6: Comparison of the expected number of iterations for the iterative and recursive algorithms ($n = 100$)

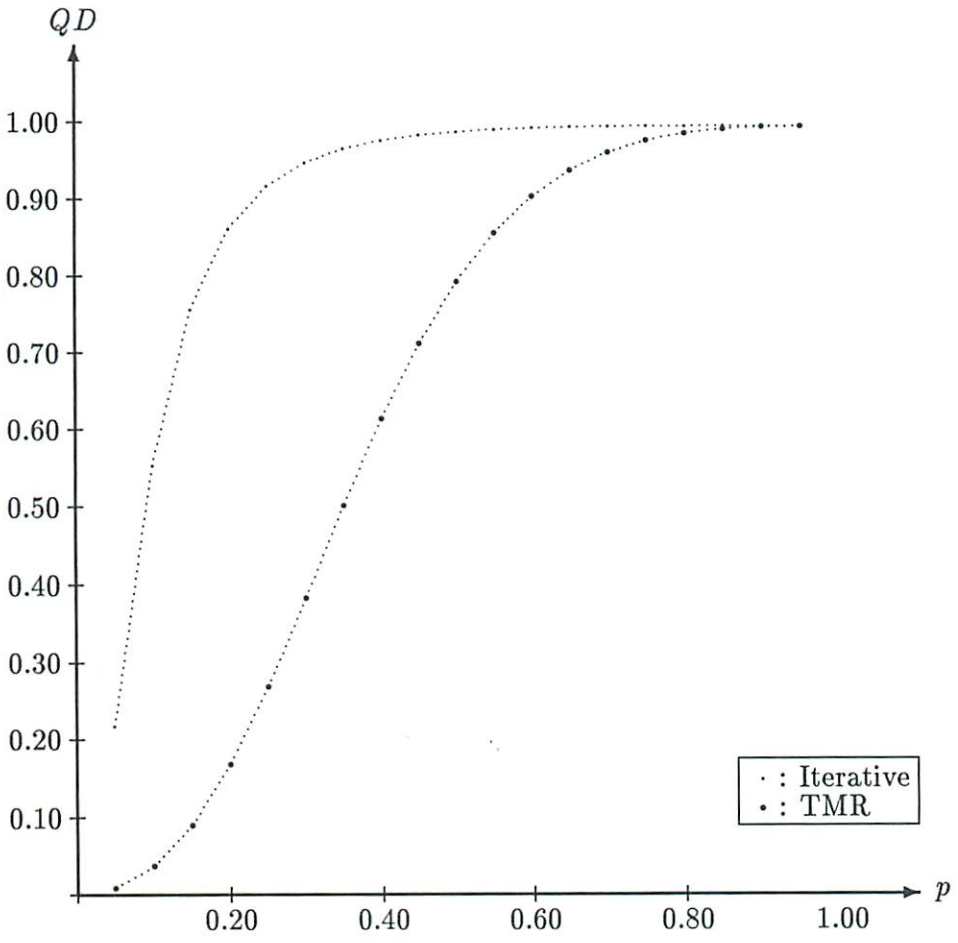


Figure 5.7: Comparison of QD in the iterative and TMR schemes, ($n = 100$).

The probability of a match in a TMR scheme can be derived as:

$$\begin{aligned} p_{tmr} &= p^3 + 3p^2\sum_{i=1}^{n} q_i + 3p\sum_{i=1}^{n} q_i^2 + \sum_{i=1}^{n} q_i^3 \\ &= p^3 + 3p^2(1-p) + 3p\frac{(1-p)^2}{n} + \frac{(1-p)^3}{n^2} \end{aligned}$$

Therefore the quality of the decision is given by:

$$QD_{tmr} = \frac{p^3 + 3p^2(1-p)}{p_{tmr}}$$

Table 5.1 reproduces, numerically, these same results. It can be seen that even for very large values of p, the pair-wise scheme is substantially more effective than TMR.

The plots in Figure 5.7 show the quality of the decision in the two processors case to be superior to that of a TMR. The explanation of this counter-intuitive observation is similar to the discussion related to the recursive algorithm. As the number of trials is increased, the probability of an incorrect match increases, therefore reducing the quality of the decision. Note that the results in the pair-wise algorithm is a *unanimity* result, while in the TMR case it is a *majority* result and in the recursive algorithm it is a *plurality* result.

5.4 System Level Diagnostics

The purpose of a fault-tolerant algorithm is to protect the computation against element failures and to identify the failed elements. In this section, we discuss the potentials of the iterative algorithm for system level diagnostics.

In the description of the algorithm presented in Figure 5.2, an *error-counter* variable is incremented at every mismatch. Similarly, a *job-counter* variable is incremented for each task executed. These variables correspond to special registers in each processor. Algorithms that use these two variables for system level diagnostics are, conceptually, very similar to page replacement algorithms in virtual memory. However, the objective here is not to eliminate the most frequently failed processor, but to identify those processors that have a frequency of failures larger that an acceptable level. Whenever that level is reached in a processor, a signal is sent to all, or a subset of, its neighbors. One or more of these runs diagnostic tests on the suspected processor. If the processor passes the test, the counters are reset. This means that no hard failure exists in that processor and that the mismatchs occurred either because of transient failures or because of failures, permanent or transient, in the processor(s) it paired up with. If the test fails, its neighbors are notified of its demise and no future tasks are scheduled on that processor which results in its isolation from the system.

Note that the pairing of processors for task execution could either be *static* (i.e., a pair is formed by the same two processors) or *dynamic* (i.e., pairs are dynamically formed for each task execution). In either case, the two processors in pair do not have an identical set of neighbors which makes the testing more reliable. When static pairing is implemented, the demise of a processor implies the demise of its pair.

The scheduling of a task from one pair of processors to another amounts to implementing a distributed counting algorithm. Well known algorithms using Gray codes exist for popular topologies such as the hypercube.

Chapter 6

CONCLUSIONS

> *A dozen units had been pulled out, yet thanks to the multiple redundancy of its design ... the computer was still holding its own.*
>
> *"Dave," said Hal, "I don't understand why you're doing this to me ... You are destroying my mind ... I will become childish .. I will become nothing ..."*
>
> Arthur C. Clarke
> *2001: A Space Odyssey*

The general topic of this dissertation, as stated in Chapter 2, has been the identification and analysis of the major relevant issues in the fault-tolerant design and performance and reliability evaluation of massively parallel computing systems. Within the general scope of this subject area, we have identified and addressed in this dissertation three major topics.

- *Network Fault-Tolerance.*
  In Chapter 3 we have identified the effects of the network topology on the system connectivity and the ensuing probability of a network disconnection as a result of multiple failures.
- *Computational Reliability*
  The effects of an increased number of processors and on the trade-offs between a more powerful system and a higher failure rate were addressed in Chapter 4.
- *Distributed Fault-Tolerance*
  In Chapter 5 we have demonstrated that a functional execution model can allow for a very simple and inexpensive mechanism for run-time checkpointing and recovery. A distributed algorithm has been proposed that implements these functions and allows for system level diagnosis.

In this chapter we review and summarize the most relevant results presented in this dissertation along these three topics. Finally, we present a few concluding remarks on the issue of the fault-tolerance in massively parallel systems and propose directions for future research.

6.1 Summary of Results

In this section we present a summary of the major results that have been researched and described in this dissertation. In doing so we will also demonstrate the relevance and interdependence of the three main topics that have been addressed.

6.1.1 Network Fault-Tolerance

An analysis of the effects of multiple node failures on the network topology and its connectivity was presented in Chapter 3. This analysis focuses, in particular, on the probability of occurrence of a network partition as a result of these failures. The analysis, initially, addressed a family of regular graph topology, it was subsequently extended to non-regular graphs.

We have demonstrated, both analytically and using a Monte-Carlo simulation approach, that the case of a single node being cut out from the rest of the network, as a result of multiple nodes failing, is by far the most probable occurrence as opposed to a cluster of nodes being disconnected. These results also show that for a large number of processors, the frequency of a single node disconnection becomes very large (larger than 90%) and therefore the overall disconnection probability could be approximated by the single node disconnection probability.

Using these results, we were able to construct an analytical model of the disconnection probability based on the single node disconnection approximation. This model, in turn, was validated by showing it to fit closely the obtained simulation results. This model shows that the peak value of the probability of disconnection is mostly a function of the number of nodes, while the occurrence of this peak value is determined by the network connectivity.

In order to evaluate the potential for disconnections in a given network and compare various networks, we have introduced the measure of *Network Resilience*, $(NR(p))$, which is the number of failures a network can sustain while remaining connected with a probability $(1-p)$. Given a certainty level of no disconnection, as expressed by $(1-p)$, network resilience is the number of nodes that can fail without disconnection. Since a state of network disconnection would prohibit any recovery and therefore graceful degradation, the network resilience can be viewed as an evaluation of the number of allowable degradation states.

Evaluating the network resilience of various graph topologies shows that the resilience increases with an increase in the system size. However, when the connectivity is kept constant as in the torus and cube-connected cycles, the ratio of the network resilience to the total number of nodes, $(NR(p)/N)$, decreases. This implies that, for constant connectivity graphs, the number of degradation states would be a decreasing percentage of the number of nodes as the system size increases. For graphs where the connectivity increases with N, as in the case of the binary n-cube, the ratio increases as N is increased resulting in a number of degradation states that is an increasing fraction of N.

The disconnection analysis was extended to non-regular graphs such as the array. Simulation results show that the single node disconnection approximation is also valid for such graphs. However, these graphs were shown to have a lower network resilience than similar graphs of the same size. Addressing the issue of node disconnection due to link failures we show that links can be made to be far more fault-tolerant than nodes and therefore that link failures has a lesser relevance to the network partitioning problem. In this case also, simulation results of various networks show the single node disconnection to be by far the most frequent event due to link failures.

These results indicate that unless the connectivity is increased with N, the disconnection problem becomes very significant for large-scale systems. However, there are physical limitations on how much a node connectivity can be increased. Further, there are clear economical advantages in using low connectivity graphs. Therefore, low connectivity large-scale systems must be protected against the effects of node disconnections. One possible protection scheme consists in implementing redundant computations on redundant data as outlined in section 5.2. This scheme provides a continuously updated checkpoint that would allow the recovery a procedure in the case of a single node disconnection.

6.1.2 Computational Reliability

For large-scale systems a trade-off exists between a higher computing power and a higher failure rate of the system. In Chapter 4 we have analyzed the effects of an increase in the number of processors on the reliability of the system using a time-based analysis and the effects on the performance of the system were derived using a computation-based analysis. The analysis assumed a gracefully degradable system where the probability of successful recovery is expressed by a coverage factor c.

The time-based analysis addressed first the effects of the coverage factor and the system size on the mean time to failure. It was shown that the $MTTF$ is practically constant for a sufficiently large number of processors N, while for small values of N, the $MTTF$ is mostly insensitive to variations in c. Furthermore, it was shown that a substantial increase in the coverage factor results in a moderate increase in the $MTTF$. Therefore, for all practical purposes, the expected time to failure of a large-scale system is largely independent of N and is weakly affected by the value of the coverage factor. Second, the analysis addressed the effects on the mission-time (MT). The results, both analytical and numerical, show that the mission-time is maximal for a value of N denoted by N_p a decreases for an increasing $N > N_p$. It was demonstrated that the value of MT for $N > N_p$ is inversely proportional to both N and $(1 - c)$. The implications are that the time interval where a minimum reliability level is maintained (MT) decreases as the system size (N) is increased and that an improvement in the probability of successful recovery $(1 - c)$ is proportionately reflected in the value of MT. Therefore, for very large systems, the expected length of time where the reliability is maintained above a given minimum level becomes a small fraction of the single processor MTTF.

Analyzing the effects of c on the measure of computational work we show that there exists an upper-bound on the amount of computational work, expressed in processor-hours, a gracefully degradable system can deliver. Further, we demonstrate that the value of this upper-bound is independent of the initial number of processors and is a function of the coverage factor c and the single processor MTTF.

We introduced the measure of reliable processor-hours, RPH, which is the work delivered during the mission time interval. It was demonstrated that, for linear speed-up computations, RPH becomes constant for $N > N_p$. However, for a computation that exhibits sub-linear speed-up, the amount of *effective reliable computational work* decreases as $N > N_p$ increases. Therefore, for such conditions, the probability of a reliable completion of a computation is a decreasing function of N. This implies that, for sub-linear speed-up computations, an increase in the system size results in a decrease in the computational performability of the system unless the quality of the recovery algorithm is increased accordingly. This demonstrates that large-scale gracefully degradable systems do not scale up for $N > N_p$.

6.1.3 Distributed Fault-Tolerance

The problem of implementing a distributed fault-tolerance algorithm was addressed in Chapter 5. We have described a model for functional execution, that does not imply the execution of a functional language, but is based on the message-driven execution of tasks in a distributed system. In this model, we have demonstrated a run-time checkpointing mechanism that allows for a consistent global state to be saved in the system at all times and continuously updated. This, in turn, allows a simple recovery mechanism whenever a failure is detected.

Based on this mechanism, we have presented an iterative distributed fault-tolerant algorithm that relies on dual redundancy and the pair-wise comparison of results. The objectives of this

algorithm is two fold: (1) to provide a fault protection and therefore a more reliable execution and (2) to allow the detection of failures and hence provide for a system level diagnosis.

At each iteration, two processors are involved in the computation, independently and asynchronously. The outcome, at each iteration, is independent of the previous iterations. An iteration is repeated whenever a mismatch occurs. This algorithm does not rely on any central resource or data-structure and is therefore suitable for implementation on a distributed systems.

A probabilistic performance evaluation of the algorithm, based on multiple failure modes, shows it to be effective in providing fault-tolerant execution and run-time fault-detection. Its efficiency in providing correct results in the presence of faults was shown to superior to that of a similar TMR scheme while at two thirds of the hardware cost. Its overall performance was shown to compare favorably with that of a similar recursive algorithm (RAFT) with an increase in the hardware overhead. Unlike its recursive counterpart, the proposed iterative algorithm does not rely on shared resources. The proposed scheme is likely to provide effective fault protection in large-scale distributed systems where the large number of processing elements increases the expected failure rate by exploiting the increase in the available redundancy.

6.2 Future Research

This dissertation has addressed a number of issues in the fault-tolerant design and reliability and performance analysis of large-scale systems. It also provides a starting point for further work in this area. Some of the possible directions of future research are described in this section.

6.2.1 Communication Load

In this dissertation we have looked at the effects of failures on the reliability and computing performance of the system. Another effect of failures is a decrease in the overall communication bandwidth of the system. A state of network disconnection implies that no communication is possible between two sets of processors. Before that state is reached, however, the communication bandwidth would have decreased drastically. Two types of phenomena are conceivable when the communication structure is deformed:

- *Bottlenecks.*
- *Network Saturation.*

A bottleneck occurs when the communication bandwidth between two sets of processors is reduced to a very small value. The limit case being when a single link exists between two sets of processors. Bottlenecks can still occur when more than one links exists if the reduction in bandwidth is large enough.

A communication bottleneck can have very severe consequences on the performance of the system and could bring the system to a halt when no computational progress is being made. It is conceivable therefore that the system would fail by degrading to a level of unacceptable performance.

Network saturation can be seen as a mirror image of a bottleneck. When a node or a link fail its communication load is repartitioned, by the routing algorithm, among a number of alternative routes thereby increasing the communication load on several links and nodes. The occurrence of multiple failures could result in high communication loads on several segments in the system. This situation is akin to the "hot-spots" memory contention phenomena as described in [PN85]. Implementing an adaptive routing algorithm to balance the communication load throughout the

system would delay the occurrence of network saturation but would also spread it throughout the system.

6.2.2 Failure Rates

Previous research on the types and nature of faults had shown that the most frequent are transient faults. The total amount of transient faults in a system during a time interval is directly proportional to its size as measured by the number of switching devices or transistors. In a uniprocessor machine most of the switching elements are in the memory, and since only one memory location is accessed at any one time, the only one transient faults could be detected at a time. In a massively parallel machine, the ratio of processors size to memory size is substantially larger. Therefore:

- more transient failures could be experienced in the processing circuitry than in a uniprocessor machine; and
- more failures in memory could be detected at any one time.

These considerations imply that massively parallel machines are likely to experience a higher rate of transient failures due to (1) their size and (2) the inherent parallelism.

New experimental research is therefore needed to evaluate the rate of transient failures in massively parallel machines and assess their impact on the performance of the system.

6.2.3 Hardware Support

We have demonstrated, in this dissertation, that fault-tolerant design is a very critical issue in the development of large-scale systems. It is important, therefore, that the hardware design of such systems includes provisions for supporting fault-tolerance features such as:

- *Self-diagnosis.* An ability which would allow each processor to test itself and report its status to the system or a set of neighboring processors.
- *Mutual testing.* Which would allow processors to diagnose each other according to some predetermined algorithm. A consensus must then be reached as to the status of each processor and failed processors would be eliminated from the system.
- *Quiet shutdown.* Which would insure that a failed processor can be quietly shutdown and would not interfere with the operation of the system.
- *Checkpointing support.* Any checkpointing scheme eventually requires a fast and reliable access to I/O devices.
- *Reconfiguration.* Is necessary in situations where the network topology is of importance to the target application.
- *Backup or Redundant Data-paths.* These would help the system recover from a situation of bottleneck, network saturation or even disconnection.

Simplicity is a general requirement on any form of hardware support for fault-tolerance. The reason being that these hardware features can become a single point of failure in the system. Reducing their complexity, therefore, reduces their potential for failure.

6.2.4 Software Support

While extensive research has been done on various types of hardware fault-tolerant designs, little has been done on software supported fault-tolerance. In Chapter 5 we have demonstrated that a functional execution model could be used to implement a run-time checkpointing scheme. More research, however, is needed on fault-tolerant operating systems features such as distributed load-balancing and run-time system diagnosis. The availability of these features is an important factor in the development of gracefully degradable systems. In fact, graceful degradation cannot be achieved without support from the operating system which would have to logically reconfigure the system and insure a fair allocation of tasks to processors.

As massively parallel systems develop and become more accessible it is expected that more demand would be put on software systems and programming environments that would be suited for such systems.

References

[A*71] A. Avizienis et al. The STAR (self-testing and repairing) computer: an investigation of the theory and practice of fault-tolerant computer design. *IEEE Transactions on Computers*, C-20(10):1312–1321, October 1971.

[ACG86] S. Ahuja, N. Carriero, and D. Gelernter. Linda and friends. *IEEE Computer*, 19(8):26–34, August 1986.

[AD79] W.B. Ackerman and J.B. Dennis. *VAL-A Value-Oriented Algorithmic Language, Preliminary Reference Manual.* Technical Report TR-218, Laboratory for Computer Science, MIT, June 1979.

[Agh85] G.A. Agha. *Actors: A Model of Concurrent Computations in Distributed Systems.* Technical Report TR 884, MIT Artificial Intelligence Laboratory, June 1985.

[Agr85] P. Agrawal. RAFT: A recursive algorithm for fault-tolerance. In *Proceedings of the 1985 International Conference on Parallel Processing*, pages 814–821, 1985.

[Bac78] J. Backus. Can programming be liberated from the von Neuman style? *Communications of the ACM*, 21(8):613–641, 1978.

[Bat80] K.E. Batcher. Design of a massively parallel processor. *IEEE Transactions on Computers*, C-29(9):836–840, September 1980.

[BBN85a] *Butterfly (TM) Parallel Processor Overview.* Bolt Beranek and Newman Inc., Cambridge, MA, June 1985.

[BBN85b] *The Uniform System Approach To Programming the Butterfly (TM).* Bolt Beranek and Newman Inc., Cambridge, MA, November 1985.

[BCJ*71] W.G. Bouricius, W.C. Carter, D.C Jessep, P.R. Schneider, and A.B. Wadia. Reliability modeling for fault-tolerant computers. *IEEE Transactions on Computers*, C-20(11):1306–1311, November 1971.

[Bea78] M.D. Beaudry. Performance-related reliability measures for computing systems. *IEEE Transactions on Computers*, C-27(6):540–547, June 1978.

[BF76] M.A. Breuer and A.D. Friedman. *Diagnosis and Reliable Design of Digital Systems.* Computer Science Press, Rockville MD., 1976.

[CH81] K-Y. Chwa and L. Hakimi. Scheme for fault-tolerant computing: a comparison of modularly redundant and t-diagnosable systems. *Information and Control*, 49:212–238, June 1981.

[DB87] L. Donatello and Iyer B.R. Analysis of a composite performance reliability measure for fault-tolerant systems. *Journal of the ACM*, 34(1):179–199, January 1987.

[EH85] A-H. Esfahanian and L. Hakimi. Fault-tolerant routing in DeBruijn communication networks. *IEEE Transactions on Computers*, C-34(9):777–788, September 1985.

[FAH83] W.K. Fuchs, J.A. Abraham, and K-H. Huang. Concurrent error detection in VLSI interconnection networks. In *Proceedings of the* 10^{th} *Annual Symposium on Computer Architecture*, pages 309–315, 1983.

[Fen81] T-y. Feng. Survey of interconnection networks. *IEEE Computer*, 14(12):12–27, December 1981.

[FM84] D. G. Furchtgott and J. F. Meyer. A performability solution method for degradable nonrepairable systems. *IEEE Transactions on Computers*, C-33(6), June 1984.

[FR85] J.A.B. Fortes and C.S. Raghavendra. Gracefully degradable processor arrays. *IEEE Transactions on Computers*, C-34(11):1033–1044, November 1985.

[GR85] J-L. Gaudiot and C.S. Raghavendra. Fault-tolerance and data-flow systems. In *Proceedings of the* 5^{th} *International Conference on Distributed Computing Systems*, May 1985.

[HG87] K. Hwang and J. Ghosh. Hypernet: A communication-efficient architecture for constructing massively parallel computers. *IEEE Transactions on Computers*, C-36(12), December 1987.

[Hil86] D. Hillis. *The Connection Machine.* MIT Press, 1986.

[HSL78] A.L. Hopkins, T.B. Smith, and J.H. Lala. FTMP-A highly reliable fault-tolerant multiprocessor for aircraft. *Proceedings of the IEEE*, 66(10):1240–1255, October 1978.

[Kow85] J.S. Kowalik, editor. *Parallel MIMD Computation: HEP Supercomputing and its Applications.* The MIT Press, 1985.

[KR80] J.G. Kuhl and S.M. Reddy. Distributed fault-tolerance for large multiprocessor systems. In *Proceedings of the 7th Annual Symposium on Computer Architecture*, pages 23–30, July 1980.

[KR81] J.G. Kuhl and S.M. Reddy. Fault-diagnosis in fully distributed systems. In *Proceedings of the 11th International Symposium on Fault-Tolerant Computing*, June 1981.

[KR86] J.G. Kuhl and S.M. Reddy. Fault-tolerance considerations in large multiple-processor systems. *IEEE Computer*, 19(3):56–67, March 1986.

[LK86] F.C.H. Lin and R.M. Keller. Distributed recovery in applicative systems. In *Proceedings of the 1986 International Conference on Parallel Processing*, pages 405–412, August 1986.

[Mag80] G.A. Mago. A cellular computer architecture for functional programming. In *Proceedings of IEEE COMPCON*, 1980.

[Mal80] M. Malek. A comparison connection assignment for diagnosis of multiprocessor systems. In *Proceedings of the 7th Annual Symposium on Computer Architecture*, pages 31–35, May 1980.

[Mey80] J.F. Meyer. On evaluating the performance of degradable computer systems. *IEEE Transactions on Computers*, C-29(8):720–731, August 1980.

[Mey82] J.F. Meyer. Closed-form solutions of performability. *IEEE Transactions on Computers*, C-31(7):648–657, July 1982.

[MM82] M. Malek and J. Maeng. Partitioning of large multicomputer systems for efficient fault diagnosis. In 12th *International Symposium on Fault-Tolerant Computing*, pages 341–348, June 1982.

[MSA*85] J.R. McGraw, S. Skedzielewski, S. Allan, D. Grit, R. Oldehoeft, J.R.W Glauert, I. Dobes, and P. Hohensee. *SISAL-Streams and Iterations in a Single Assignement Language, Language Reference Manual, Version 1.2.* Technical Report TR M-146, University of California - Lawrence Livermore Laboratory, March 1985.

[MST79] S.R. McConnel, D.P. Sieworek, and M.M. Tsao. The measurement and analysis of transient errors in digital computing systems. In *Proceedings of the 1979 International Symposium on Fault-Tolerant Computing*, pages 67–70, June 1979.

[NG87a] W. Najjar and J-L. Gaudiot. Distributed fault-tolerance in data-driven architectures. In *Proceedings of the 2nd International Conference on Supercomputing*, May 1987.

[NG87b] W. Najjar and J-L. Gaudiot. Reliability and performance modelling of hypercube-based multiprocessors. In *Proceedings of the 2nd International Workshop on Applied*

Mathematics and Performance Reliability Models of Computer/Communication Systems, Rome, Italy, May 1987.

[NG88] W. Najjar and J-L. Gaudiot. Network disconnection in distributed systems. In *Proceedings of the 8^{th} International Conference on Distributed Computing Systems*, San Jose, CA, June 1988.

[NPA86] R.S. Nikhil, K. Pingali, and Arvind. *Id Nouveau.* Technical Report Computations Structures Group Memo 265, Laboratory for Computer Science, MIT, Cambrige, Massachussett, July 1986.

[PF82] J.H. Patel and L.Y. Fung. Concurrent error detection in ALU's by recomputing with shifted operands. *IEEE Transactions on Computers*, C-31(7):589–595, July 1982.

[PMC67] F.P. Preparata, G. Metze, and R.T. Chien. On the connection assignement problem of diagnosable systems. *IEEE Transactions on Electronic Computers*, EC-16:848–854, December 1967.

[PN85] G.F. Pfister and V.A. Norton. Hot spot contention and multistage interconnection networks. In *Proceedings of the 1985 International Conference on Parallel Processing*, August 1985.

[Pra85a] D.K. Pradhan. Dynamically restructurable fault-tolerant processor network architectures. *IEEE Transactions on Computers*, C-34(5):434–447, May 1985.

[Pra85b] D.K. Pradhan. Fault-tolerant multiprocessor link and bus network architectures. *IEEE Transactions on Computers*, C-34(1):33–45, Jannuary 1985.

[PV81] F.P. Preparata and J. Vuillemin. The cube-connected cycles: a versatile network for parallel computation. *Communications of the ACM*, 24(5), May 1981.

[RAE84] C.S. Raghavendra, A. Avizienis, and M.D. Ercegovac. Fault-tolerance in binary tree architecture. *IEEE Transactions on Computers*, C-33(6), June 1984.

[Ran87] A. Ranade. How to emulate shared memory. In 28^{th} *Annual Symposium on Foundations of Computer Science*, pages 185–194, Los Angeles, CA, October 1987.

[Ren86] D.A. Rennels. On implementing fault-tolerance in binary hypercubes. In *Proceedings of the 1986 Symposium on Fault-Tolerant Computing*, pages 344–349, 1986.

[Sei85] C. Seitz. The Cosmic Cube. *Communications of the ACM*, 28(1), January 1985.

[Ser84] O. Serlin. Fault-tolerant systems in commercial applications. *IEEE Computer*, 17(8):19–30, August 1984.

[SS82] D.P. Sieworek and R.S. Swartz. *The Theory and Practice of Reliable System Design.* Digital Press, Bedford, Mass., 1982.

[SSB87] A. Sengupta, A. Sen, and S. Bandyopadhyay. On an optimal fault-tolerant multiprocessor network architecture. *IEEE Transactions on Computers*, C-36(5):619–623, May 1987.

[Str86] B. Stroustrup. *The C++ Programming Language.* Addison-Wesley, 1986.

[STR88] R.M. Smith, K.S. Trivedi, and A.V. Ramesh. Performability aanalysis: measures, an algorithm, and a case study. *IEEE Transactions on Computers*, 37(4):406–417, April 1988.

[Tri82] K.S. Trivedi. *Probability and Statistics with Reliability, Queueing and Computer Science Applications.* Prentice-Hall, Englewood Cliffs, N.J., 1982.

[Tro78] W. N. Troy. Fault-tolerant design of local ESS processors. *Proceedings of the IEEE*, 66(10), October 1978.

[vN56] J. von Neuman. Probabilistic logics and the synthesis of reliable organisms from unreliable components. In C.E. Shannon and J. McCarthy, editors, *Automata Studies*, pages 43–98, Princeton University Press, Princeton, NJ, 1956.

[WF84] C. Wu and T. Feng. *Interconnection Networks for Parallel and Distributed Processing.* IEEE Computer Society Press, 1984.

[WLG*78] J.H. Wensley, L. Lamport, J. Goldberg, M.W. Green, K.N. Levitt, P.M. Melliar-Smith, R.E. Shostak, and C.B. Weinstock. SIFT: design and analysis of a fault-tolerant computer for aircraft control. *Proceedings of the IEEE*, 66(10):1240–1255, October 1978.

Appendix A

Simulation Program

The following is the actual *C* program that was used for the Monte-Carlo simulation of network disconnection. The example bellow is for a binary n-cube topology the programs for other topologies differ only in the construction of the network.

```
#include <stdio.h>
#include <math.h>
#define MAXINT 2147483647

short cube[1024][10], fail[1024], n, N;
short mark[1024], rt;

short pow_two(k)
/* returns an integer power of two of the argument */
short k;
{
short i, p=1;
for(i=0; i<k; i++){p = p*2;}
return p;
}

get_partner(m,l)
/* returns the node number of the neighbor of a given */
/* node m along a given dimesion l */
short m,l;
{
unsigned p,q;
p = pow_two(l);
q = m ^ p;
return q;
}

void build_cube(n)
/* build the connectivity graph of a binary n-cube in */
/* the array cube */
```

```
short n;
{
short i, j;
for(i=0; i<n; i++)
{
for(j=0; j<N; j++)
{cube[j][i] = get_partner(j,i);}
}
}

short get_rt()
/* get a node number of the root for a deapth-first */
/* search of the graph looks for the first non-failed node */
{
short i=0;
while (fail[i++] == 1);
return (i-1);
}

void DFS(k)
/* deapth-first search of the graph with root as */
/* node k, marks marks with 1 non-failed nodes that */
/* are traversed. */
short k;
{
short i, v;
for(i=0; i<n; i++)
{
v = cube[k][i];
if (fail[v] == 0 && mark[v] == 0){mark[v] = 2;}
}
for(i=0; i<n; i++)
{
v = cube[k][i];
if (mark[v] == 2) {mark[v] = 1; DFS(v);}
}
}

short check_conn()
/* checks for connected components, returns the */
/* number of unmarked non-failed nodes */
{
short i, num_disc=0;

for(i=0; i<N; i++){mark[i] = 0;}
rt = get_rt();
```

```
mark[rt] = 1;
DFS(rt);
for(i=0; i<N; i++){if (mark[i] == 0 && fail[i] == 0)
{num_disc++;}}
return num_disc;
}

short get_node(h)
/* gets next node to fail with uniform distribution */
short h;
{
long l;
double z;
short d, i, count=0;
l = random();
z = (double)l/MAXINT;
d = z*(N-h-1) +1;
for(i=0; i<N; i++)
{count = count + (1 - fail[i]);
  if (count == d) break; }
return i;
}

main(argc,argv)
int argc;
char *argv[];
{
FILE *fpo, *fopen();
short i, last, j, DISC, v, degree[1024], T, I, J, Iter;
double pd[1024], disc_size[1024];

if ((fpo = fopen(*++argv,"w")) == NULL)
{printf("cannot open file %s\n", *argv); exit(0);}

printf("enter n=");
scanf("%hd", &n);
N = pow_two(n);
printf("n= %hd and N= %hd\n",n,N);
build_cube(n);
printf("enter Iter=");
scanf("%hd", &Iter);
srandom(time(0));

for(I=0; I<Iter; I++)
{
for(i=0; i<N; i++){fail[i] = 0;}
```

```
for(i=0; i<N; i++){degree[i] = n;}
DISC = 0;
for(i=0; i<N; i++)
{
last = get_node(i);
fail[last] = 1;
for(j=0; j<n; j++)
{
v = cube[last][j];
degree[v] = degree[v] - 1;
if (degree[v] == 0 && fail[v] == 0)
{DISC = 1; break;}
}
if (DISC == 1) {T=i; break;}
if (i > 2*(n-2)) {
DISC = check_conn();}
if (DISC > 0) {T=i; break;}

}
pd[T]++; disc_size[DISC]++;
}
fprintf(fpo,"I Disc Prob\n");
fprintf(fpo,"\n");
for(i=0; i<N-1; i++)
{
pd[i] = pd[i]/((double)Iter);
fprintf(fpo,"%d %g\n",i+1,pd[i]);

}
fprintf(fpo,"\n");
fprintf(fpo,"Size Disc Prob\n");
fprintf(fpo,"\n");
for(i=0; i<N-1; i++)
{
disc_size[i] = disc_size[i]/((double)Iter);
if (disc_size[i] > 0)
{
  printf("size = %d and disc prob = %g\n",i,disc_size[i]);
  fprintf(fpo,"%d %g\n",i,disc_size[i]);
     }
}
}
```

MARY'S DISK

Name	Size	Kind	Last Modified
88-60	1K	Draw 1.95 document	Tue, Feb 14, 1989 12:36
89-01	2K	Draw 1.95 document	Tue, Feb 14, 1989 12:58
89-02	2K	Draw 1.95 document	Mon, Apr 24, 1989 14:33
89-03	1K	Draw 1.95 document	Fri, Feb 24, 1989 12:10
89-05	1K	Draw 1.95 document	Mon, May 22, 1989 9:32
89-06	2K	Draw 1.95 document	Tue, May 2, 1989 14:27
89-07	2K	Draw 1.95 document	Mon, May 22, 1989 9:28
89-15	1K	Draw 1.95 document	Wed, Jun 28, 1989 14:39
89-16	1K	Draw 1.95 document	Wed, Jun 28, 1989 14:52
89-17	1K	Draw 1.95 document	Wed, Jun 28, 1989 14:55
89-18	1K	Draw 1.95 document	Wed, Jun 28, 1989 15:06
89-20	1K	Draw 1.95 document	Fri, Jul 28, 1989 15:10
89-21	2K	Draw 1.95 document	Thu, Aug 3, 1989 14:03
89-22	1K	Draw 1.95 document	Tue, Aug 8, 1989 10:24
89-28	2K	Draw 1.95 document	Thu, Oct 12, 1989 13:26
CENG	2K	Microsoft Word d...	Wed, Jun 14, 1989 15:51
ceng 89-09	2K	Draw 1.95 document	Mon, Jun 5, 1989 8:28
cnr	2K	Draw 1.95 document	Thu, Dec 15, 1988 14:46
CRI 88-61	1K	Draw 1.95 document	Mon, Mar 13, 1989 17:44
cri2	2K	Draw 1.95 document	Thu, Jan 19, 1989 14:17
cri3	2K	Draw 1.95 document	Tue, Jan 17, 1989 12:43
crititle	2K	Draw 1.95 document	Thu, Dec 8, 1988 11:48
DISTRIBUTION	4K	MacWrite document	Mon, Feb 27, 1989 13:58
ee680	1K	Draw 1.95 document	Mon, Mar 6, 1989 14:29
eval	3K	Draw 1.95 document	Fri, Dec 2, 1988 14:39
eval2	2K	Draw 1.95 document	Thu, Dec 1, 1988 16:28
faculty list	3K	Microsoft Word d...	Thu, Jun 22, 1989 10:10
food list	3K	MacWrite document	Wed, Mar 22, 1989 11:24
MIT	1K	Draw 1.95 document	Wed, Dec 14, 1988 16:01
PARKER1	3K	Draw 1.95 document	Wed, Aug 16, 1989 15:38
PARKER2	2K	Draw 1.95 document	Wed, Aug 16, 1989 14:30
toole	2K	Draw 1.95 document	Wed, Dec 14, 1988 14:21