# MULTICOMPUTER NETWORKS FOR SMART STRUCTURES

by

John T. McHenry

Dissertation submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Electrical Engineering

APPROVED:

Scott F.

Dr. S.F. Midkiff, Chairman

Marc Abrams

Dr. M. Abrams

Richard O. Claus

Dr. R.O. Claus

Dr. N.J. Davis

Harley H Cudney

Dr. H.H. Cudney

June, 1993

Blacksburg, Virginia

# MULTICOMPUTER NETWORKS FOR SMART STRUCTURES

by

John T. McHenry

Dr. S. F. Midkiff, Chairman

Electrical Engineering

## (ABSTRACT)

A crucial element of a smart structure is the computer system that processes data collected by sensors and determines an appropriate response. Multicomputers possess many capabilities that are required in computer systems for smart structures. This research examines the implementation and use of multicomputers for distributed processing in smart structures.

The research begins by examining previous research and showing the suitability of multicomputers for distributed processing in smart structures. Appropriate cost and performance metrics for evaluating multicomputer architectures are defined. The cost metrics are the number of processors, the number of communication links, and the length of fiber required to embed the network in the structure. The performance measures are the algorithm cycle time and the mean and standard deviation of message latency in the network. The scalability of these metrics is also examined. A key issue in the examination of these metrics is how their application to smart structures differs from their application in traditional systems.

The research continues by using a three-processor testbed network to identify general characteristics of algorithms that may be executed in smart structures. The testbed network uses fiber optic sensing, the MIL-STD-1773 communication protocol, and several different assignments for partitioning the necessary computations among the processing nodes to determine the shape of a triangular structure. The effects of math coprocessing on performance and the viability of hybrid links, in which a single optical fiber is used simultaneously for sensing and communication, are also demonstrated.

Simulation models of a damage detection, location, and estimation algorithm implemented in VHDL, a hardware description language, are used to examine and compare the performance of multicomputer interconnection network topologies. The topologies examined in this research are a binary hypercube, a custom planar topology, and a custom hierarchical topology. The ability of hierarchical architectures to limit cost while providing acceptable performance is demonstrated. The simulations also examine the effects of background message traffic and the ratio of communication time to processing time on performance. The combined results of the testbed and simulation experiments show the importance of process assignment and scheduling.

# Acknowledgements

I would first like to thank the people who directly contributed to the completion of this dissertation. This thanks begins with the acknowledgement of Dr. Scott F. Midkiff for the guidance he provided during the course of this research and the many hours he spent editing preliminary versions of this paper. I also thank Dr. Marc Abrams, Dr. Richard Claus, Dr. Harley Cudney, and Dr. Nathaniel Davis for serving on my committee, Dr. Douglas Lindner for discussing the damage detection algorithm, and Russell May for his help in constructing the fiber optic network. Furthermore, I would like to acknowledge that this work was supported in part by the generous financial support of Mrs. Marion Via to the Bradley Department of Electrical Engineering at Virginia Tech.

A large degree of emotional support and encouragement has been received from the many friendships I have made while at Virginia Tech. Although the list is too long to mention, several names deserve specific recognition. Jeff Allen, Dan and Karen Greenhaus, Scott Harrison, as well as innumerable other associates of the Sigma Alpha Fraternity, have shared in many of the moments that made my years at Virginia Tech memorable.

I would also like to thank Chris Waltz and the rest of Tundra 212 for the many hours I spent listening to their music. I know she would not want to be listed so close to the band, but I also have to acknowledge Kristen Waters for the hours of endless conversation this year that helped me to remain focused on my educational goals.

Lastly, and most importantly, no list of thanks would be complete without recognizing my parents and family for their continued support of my educational endeavors.

# Table of Contents

# List of Figures

# List of Tables

# Chapter 1. Introduction

A smart material is defined to be a physical fabric into which sensing, actuation, control, and signal processing functions are integrated [1]. Smart structures are physical structures that are constructed using smart materials and are able to sense and react to their environment. The function of a smart structure, including the phenomena to be sensed and the manner in which the structure should react, are determined by the particular application.

A crucial element of a smart structure is the computer system that processes data collected by sensors and determines an appropriate response. Multicomputers possess many capabilities that are required in computer systems for smart structures. Multicomputers, also known as distributed memory computers, are a type of parallel processing system that consist of multiple processing nodes connected by a communications network. Each node has computing, memory, and communication resources. The computing resources perform the processing tasks assigned to the node. The memory stores both program and data. The communication facilities transmit and receive data from other processing nodes. Multicomputer architectures can support a variety of applications.

Many applications for smart structures have been proposed. Embedded sensors can be used to perform nondestructive evaluation during the fabrication of composite materials. Sensors and actuators in space structures can be used to detect and damp out vibration. Applications have been proposed for aircraft in which smart structure technology is used to monitor the physical health of the aircraft and thus eliminate the need for costly manual maintenance inspections. Also, the maneuverability and dynamic stability of an aircraft in flight may benefit from smart structure techniques.

The motivation for smart structures arises from the potential benefits of composite materials and fiber optic sensing technologies. Early literature describes advances occurring in these areas without specific regard for the system-level implementation. A better understanding of the motivating technologies of smart structures has been gained that allows researchers to suggest computational requirements for some structures, but the computer and communication technologies that are needed to fulfill these requirements have been largely neglected in research on smart structures. Multicomputer architectures must be developed to fulfill the processing requirements of smart structures and, simultaneously, the computational requirements of smart structures must be tailored to the capabilities of multicomputer architectures. Thus, it is now both feasible and necessary to examine communication and computation in smart structures.

The overall objective of this research is to examine the effectiveness of multicomputer-based distributed processing in smart structures. Initially, the suitability of multicomputers for distributed processing in smart structures is determined. The next goal is to define appropriate cost and performance metrics for evaluating multicomputer architectures for smart structures. The third goal of the research is to identify general characteristics of the algorithms that may be executed in smart structures. Issues such as the types of calculations that the algorithms are likely to perform and the effect of the ratio of communication time to processing time on algorithm execution are examined. The final goal of this research is to determine a set of characteristics that interconnection network topologies for smart structures should possess.

The objectives of this research are achieved in three ways. First, background research concerning smart structures is used to determine the cost and performance metrics. A three-processor testbed network is then used to examine several different organizations of a shape identification algorithm, and computational characteristics of the algorithm are identified. The cost and performance metrics and the algorithm characteristics identified in the early stages of the research are applied to a simulation of a larger structure. The simulation experiments examine several different interconnection network topologies and compare their performance for a number of operating conditions. Through these methods, the research identifies desirable characteristics of multicomputer interconnection network topologies and provides insight into how to exploit these characteristics in smart structures.

Several of the cost metrics examined in this research have been studied by other researchers. These include evaluating the total number of links in the network and the number of processing nodes required. This research, however, examines these factors in the context of smart structure applications as well as other cost factors that are not commonly associated with multicomputers. Optical communication fibers increase the physical complexity of the structures in which they are embedded, and connectors for embedded fibers are difficult to implement. Thus, this research includes the amount of fiber that must be embedded in the structure as a cost factor. The scalability of several implementations is also examined. Scalability indicates how the cost of the structure changes as the size of the structure is increased.

The performance metrics that are examined are the message delay in the communication system and the cycle time of the algorithms. Metrics critical to real-time operation, such as the mean message delay, the variance of message delay, and the ability of the network to meet communication deadlines, are considered. The time it takes for an application to complete a single algorithm cycle provides insight into the interaction between the communication network and the processing nodes of a multicomputer. A key issue in evaluating both the cost metrics and performance measures is to identify how their application in smart structures differs from their application in traditional environments.

To this point, details for implementing multicomputers in smart structures have not been examined in prior work on smart structures, and the requirements of smart structures have not been addressed in multicomputers. The research presented in this paper is a first step towards developing multicomputer architectures that are optimized for smart structures. Multicomputer implementations are shown to be well suited for processing the spatially distributed sensor and actuator signals generated in smart structures. The cost and performance metrics for smart structure multicomputers are clearly defined and their relevance to smart structures is discussed. These metrics consider the real-time processing requirements of algorithms for smart structures. The manner in which factors such as the ratio of communication time to processing time affect the execution of algorithms is also discussed. The benefits of hierarchical topologies, with rich connectivity in small localized regions and additional higher level communication links that connect these regions, for executing these algorithms are also presented. A final contribution of this research is the demonstration of a hybrid scheme in which a single optical fiber is used simultaneously for sensing and communication.

Chapter 2 reviews literature that is relevant to the research. The supporting technologies that comprise smart structures are discussed. Additional research that discusses the capabilities and terminology of multicomputers is also presented. The specific objectives and methodology of the research are described in Chapter 3. Chapter 4 discusses a testbed in which a three-processor system monitors the physical shape of a triangular beam structure subjected to external forces. A damage detection, location, and estimation (DLE) algorithm is then presented that provides an example of an algorithm executed on a large smart structure. Chapter 5 describes the specific details of the algorithm and its implementation, while Chapter 6 presents simulation results obtained through a simulation model of the algorithm. Conclusions drawn from the results of the testbed example and the simulation experiments are presented in Chapter 7.

# Chapter 2. Literature Survey and Background

This chapter examines existing literature relevant to computing and communication in smart structures. The chapter begins by defining smart structures and discussing proposed smart structure applications. It then discusses the sensors and actuators that are available for smart structure applications and examines control methodologies that may be employed. The processing and data communication requirements of smart structures are identified. Also, previous research that examined computing for smart structures is discussed. The chapter continues by introducing multicomputers. In addition to defining multicomputers, this introduction examines performance issues related to both processing and communication in a multicomputer network. The chapter concludes with a discussion of how multicomputers can be used to meet the processing needs of smart structures.

## *2.1. Smart Structures*

A smart material is defined to be a physical fabric into which sensing, actuation, control, and signal processing functions are integrated [1]. Smart structures are physical structures that are constructed using smart materials and are able to sense and react to their environment. The function of the smart structure, including the phenomena to be sensed and the manner in which the structure should react, are determined by the application.

### 2.1.1. Proposed Smart Structure Applications and Related Terms

Before examining the issues of smart structures in detail, it is first necessary to understand the goals of smart structures. Claus [1] briefly describes trends in smart structure research. He predicts the development of smart structures not only for aerospace applications, but also for hydrospace and civil structures. Claus contends that the future of smart structures will be an evolution through which the various pieces of the smart structure concept, which will eventually require multi-element signal processing, are brought together into an integrated framework. The benefits of smart structures are foreseen to be the creation of materials with a greater usefulness, longer life span, and reduced manufacturing cost.

Talat [2,3] contends that the definition of smart structures is not universal. He defines the concept of "smart skins," which is closely related to smart structures, as the sensing mechanism of the structure. Thus, smart skins only detect the system's state and environment and are likened to the human nervous system. The computation in a smart skin includes only signal processing, and it identifies the presence of stimulus. "Smart structures," on the other hand, include the entire framework through which the structure first senses its current environment and state, and then takes action based on its findings. With this difference understood, Talat continues with issues that concern the smart skin concept. From his discussion, the difference in the processing requirements of smart structures and smart skins can be noted.

Talat primarily discusses smart skins for health monitoring in aircraft. He contends that aircraft maintenance procedures need to be upgraded. Aircraft inspections are labor intensive and require an interruption in the vehicle's service to be performed. Talat discusses the benefits of embedding sensors into the structure. For periodic monitoring, an interface unit could read the current state of the structure. This check should be designed so that it can be performed quickly and with minimal cost. Such a test would reduce the downtime of aircraft due to maintenance while at the same time increasing safety.

In addition to ground maintenance, Talat also sees the evolution of smart structures into military applications. The smart skin of a structure could sense in-flight damage in real-time and adjust the flight of the plane to compensate for the damage sustained. Such operation, contends Talat, will only be achievable if the information received from sensors in the aircraft can be efficiently processed.

Measures has also described applications for and development of smart structures [4,5,6]. Measures uses different terminology than Talat. He defines "passive smart structures" to be structures with an integrated system for sensing the state of the structure and its environment. This is similar to Talat's "smart skins." A "reactive smart structure" is defined to be a structure that cannot only sense its state and environment, but can also attempt to change its state in reaction to its sensor findings. A final term defined by Measures is "intelligent structures," which are structures that demonstrate an adaptive learning capability.

Measures refers to many of the same aerospace applications for smart structures as Talat. These include health monitoring, damage detection, and non-destructive evaluation of materials. In addition, Measures suggests other applications such as monitoring pipelines and storage tanks. In each of these applications, Measures goes beyond the aspects of passive structures and discusses the capabilities of reactive structures.

Mazur, *et al.* [7] discuss the role of smart structures in keeping pace with the evolution of aircraft performance. They claim that the higher performance sought by aircraft designers is achieved at the cost of aerodynamic stability in the design. The continuation of this trend requires that algorithms and systems be developed that can sense the flight of the aircraft and cause subtle changes in the flight environment. The degree of sensitivity of these flight constraints are reaching the point where they are no longer within the capabilities of a human pilot.

Also noted by Mazur, *et al.* is how the evolution of aircraft design has lead to the need for smart skins. In the early days of aircraft design, it was sufficient to over design the system and use visual inspections to perform structural maintenance. To increase performance, it was often necessary to reduce the safety margins of the design. Such a reduction, however, was not possible unless maintenance capabilities were upgraded to ensure that the aircraft were structurally sound. The introduction of smart skins to monitor the structural health of the aircraft may help to lessen the minimum safety margin required.

In addition to changes in aircraft design, there has also been a change in the average lifetime of commercial airplanes. In 1980, the active service life of an airplane was eight years, while in 1989, the active service life increased to twelve years [8]. Further increases in this lifetime are expected, while at the same time the safety of air travel cannot be sacrificed. The safety of aging airplanes could be enhanced by using smart structures to detect structural problems before they lead to catastrophic failures.

Another application driving the evolution of smart structures is the need for effective monitoring of composite materials [9,10]. The development of composites depends on the capability to test the structural integrity. In this way, areas of strain can be identified and processing methods for composites can be evaluated.

Vengsarkar and Murphy [11] discuss a three-step process that describes the smart structure's function at the birth and during the lifetime of an aircraft. During fabrication, the sensors guide the manufacturing process and detect defects. The in-service operation of smart structures includes real-time monitoring and control of the airplane's aerodynamic stability. The third step is the ability of the airplane to detect excessive material fatigue that may lead to a catastrophic failure. Thompson, *et al.*, [12] extend the role of smart technologies during the birth, life, and death cycles of structures into general applications.

The papers cited above highlight applications of smart structures. Additional papers, including [13-18], discuss the applicability of smart structures to avionic systems. Although these papers substantiate interest in smart structures, they do not add any significantly different viewpoints.

### 2.1.2. Sensors and Actuators

Smart structures use sensors to determine the current state of the structure and actuators to change the state. Thus, sensors are the primary data sources in smart structures and actuators are the primary destination for outputs. Udd [19] discusses envisioned smart structures that may require thousands of sensors. An overview of the devices that have been proposed for sensors and actuators is given below. Although passive damping techniques may also be used in smart structures, they do not require computing and are not included in this discussion. Also, fiber optic sensing is used in a testbed network that has been created for this research. Thus, the discussion of fiber optic sensors is expanded to provide a proper background for understanding this work.

#### 2.1.2.1. Fiber Optic Sensing in Smart Structures

Although fiber optic technology is commonly discussed with respect to communications, its sensing abilities have also been examined. This section discusses issues that relate to the use of fiber optic sensors in smart structures. These issues include fiber optic sensor measurements and the implementation of fiber optic sensors.

#### 2.1.2.1.1. Fiber Optic Sensor Measurements

The ability of fibers to sense a variety of phenomena is well known [6,11,20]. This ability, however, is gained not by using one technique, but rather by exploiting different mechanisms that fibers use to transmit optical energy. Measurements from these techniques require varying amounts of processing before they can be utilized by a control or health monitoring application. This section describes different sensing mechanisms and characterizes the required processing.

##### 2.1.2.1.1.1. Intensity-Based Sensors

Intensity-based sensors are among the least expensive sensors, but they also have low sensitivity [20]. They consist of a constant intensity source, a sensing mechanism that introduces loss into the signal in proportion to some sensed phenomenon, and a detector that measures the intensity of light beyond the sensing region.

Extrinsic sensors are one type of sensor that can use intensity modulation [20]. In extrinsic sensors, light exits the fiber, undergoes some form of modulation, and then reenters the fiber. One example of such a sensor is the air-gap sensor shown in Figure 2.1. A fiber is cut into two pieces and one end of each piece is placed in a hollow glass tube with a small gap between the ends of each fiber in the middle of the tube. Each piece of fiber is then attached to the structure just after it exits the tube. As the structure bends, the distance between the two fibers in the tube changes. As the gap becomes larger, less optical energy is coupled from the output of one fiber to the input of the other fiber. Thus, after the amount of energy coupled between the fibers is known for different gap distances, it is possible to determine the size of the gap from the intensity reading of the detector.

Before using the sensor, the functional relation between the sensed phenomenon and the loss in the fiber must be modeled. After this relation is known, each sensor measurement requires only a digital-to-analog (D/A) conversion of the intensity reading at the detector and a

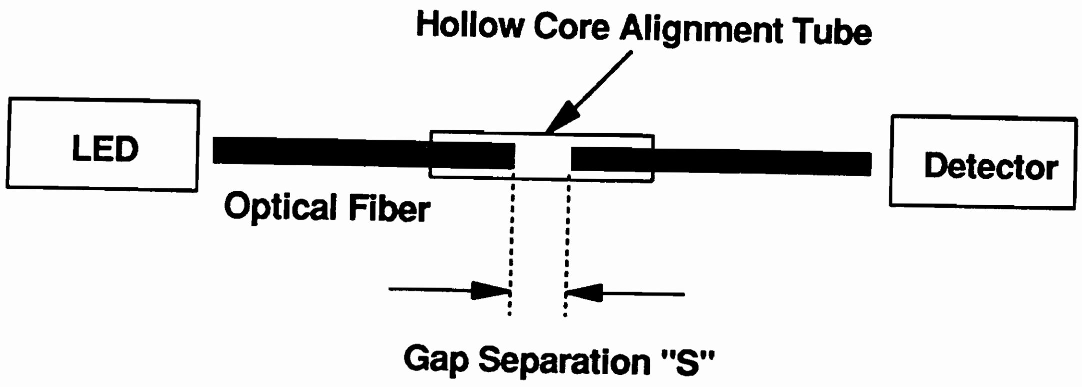


Figure 2.1. Air-gap sensor

single calculation of the function that relates the intensity reading to the sensed phenomenon. Thus, the complexity of this calculation depends on the mathematical relationship between the sensed function and the plate displacement.

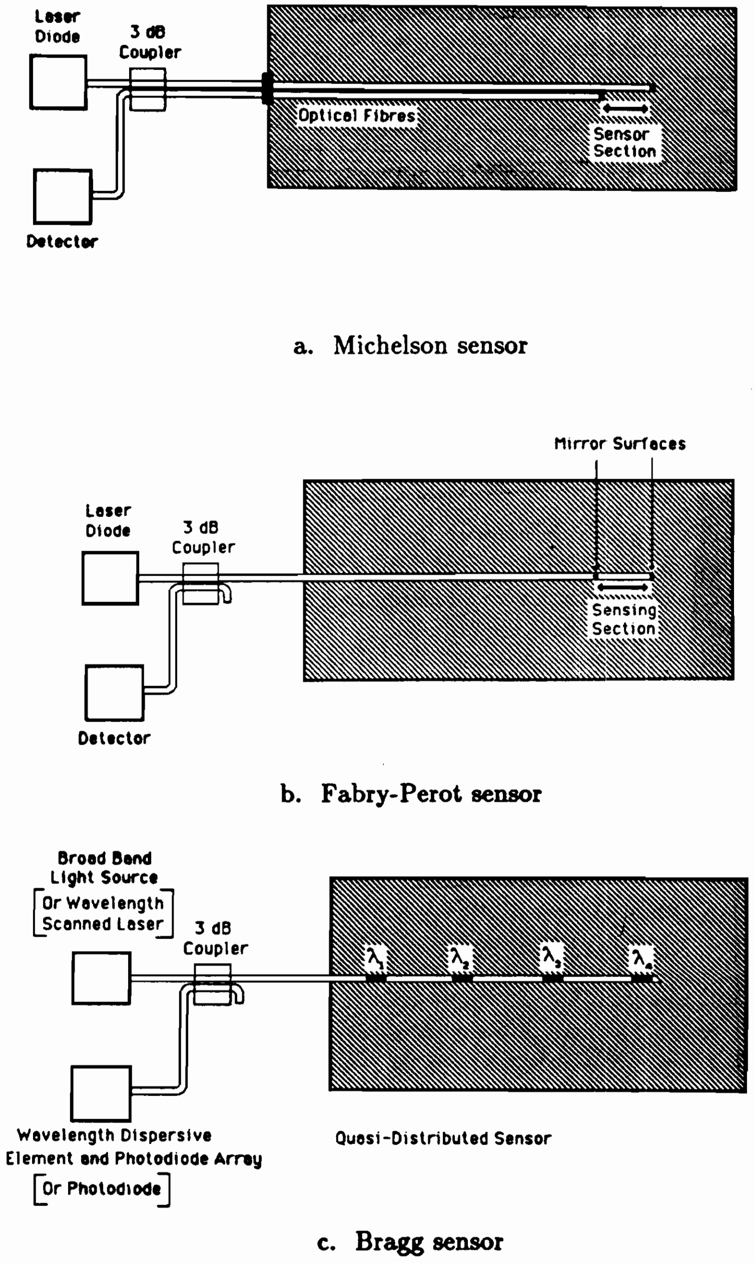
2.1.2.1.1.2. Interferometric Sensors

Interferometric sensors are a class of sensors that detect the phase change in the optical energy in a single mode fiber. A desirable characteristic of interferometric sensors is their high sensitivity. Measures states that the interferometric sensors that have the greatest potential in smart structure applications are Michelson, Fabry-Perot, and Bragg sensors [6].

The Michelson and Fabry-Perot sensors, shown in Figure 2.2a and Figure 2.2b, respectively, couple energy into one end of a fiber and reflect this energy at the other end [21]. In both of these sensors, the optical energy is permitted to follow two different paths as it propagates down the fiber. The energy for each path is coupled onto a single fiber just before it is received at the detector, and the phase difference between the two paths is determined. As the fiber is subjected to varying strain and temperature, the length of the optical path changes. This change can be detected as a variation in the phase difference of the two signals.

Before using these sensors, a function that relates the change in the path length to the sensed phenomenon is determined. After examining the phase difference, and thus the change in the path length, this predetermined function is used to generate the sensor reading. The computational complexity associated with interferometric sensors, however, is not a result of the function that relates the change in path length to the sensed phenomenon. Instead, the dominant factor in the computational complexity is the signal processing that is required to calculate the phase change.

A Bragg sensor is shown in Figure 2.2c. These sensors are similar to the Fabry-Perot and Michelson sensors in that the detected light pulse is reflected at one end of the fiber. A series of



a. Michelson sensor

b. Fabry-Perot sensor

c. Bragg sensor

**Figure 2.2. Interferometric sensors [6]**

periodic variations in the index of refraction of the glass used in the core of the fiber are created that cause Bragg reflections at set wavelengths. The sensing regions are the areas in which the refractive index is altered, and as the environment of these regions changes, the frequency that they reflect is also affected. By examining changes in reflected frequencies, it is possible to determine what is happening in each of the sensing regions. The amount of data processing that is required depends on how much of the frequency identification can be performed within the optical hardware, but in general, Bragg sensors and Fabry-Perot sensor require a comparable level of processing [22].

2.1.2.1.1.3. Modalmetric Sensors

Each mode travelling in a fiber has a different modal frequency and phase velocity. These properties of the modes can be exploited to provide sensing. Modal domain sensing uses the interference between two modes travelling in the fiber as the sensing mechanism. A major difficulty with these sensors is the inability to isolate the sensing region of the fiber.

Cox and Lindner [23] describe the use of a modal domain sensor to sense vibration in a clamped beam. Two optical modes from a monochromatic light source are coupled into an elliptical core fiber and their interference causes two intensity lobes to appear on the cross-section of the sensing region of the fiber. The intensity in each of the lobes is determined by the coupling of the two optical modes that propagate in the fiber. This coupling, in turn, is affected by the integral of the axial strain on fiber, and thus straining the fiber causes the intensity of one of the lobes to diminish while the intensity of the other lobe increases. At the end of the sensing region of the fiber, an offset fusion splice is used to couple the energy of one of the lobes into an insensitive lead-out fiber. The energy of the selected output lobe can be determined by examining the intensity of light in the lead-out fiber. Equations that relate the intensity pattern of the output to the axial strain are then used to determine the integral of the axial strain in the fiber. Using this technique, some of the signal processing is performed by the spatial filter that is created at the splice. Thus, a processor only has to calculate the equation that relates intensity to strain.

One disadvantage of this technique is that even though the axial strain is measured along the length of the fiber, the final sensor measurement is a scalar quantity. Weighted modal domain sensors have been proposed as a method for providing a distributed sensor measurement on a single fiber [24,25]. These sensors exploit intrinsic characteristics that can be manufactured into the fiber to vary the coupling characteristic of the two modes along the length of the fiber. In this way, distributed measurements of the axial strain can be made using a single fiber. The complexity of computation required depends on the coupling equations that are determined when the fiber is manufactured.

#### 2.1.2.1.2. Implementation Concerns

Many details must be considered in selecting the best sensor for a given application. It is necessary to understand how the implementation of fiber optic sensors affects a smart structure. The following paragraphs discuss the advantages and disadvantages of fiber optic sensors.

A primary benefit of fiber optic sensors is their light weight. Fiber optic sensors are often made up of two regions, the sensing region and a feedthrough region that propagates the effects of the sensing region to a detector. The light weight of the fibers themselves in each of these regions, as compared to the weight of piezoelectric film sensors and electrical transmission wires, reduces the weight of the sensor. Also, fibers are immune to electromagnetic interference (EMI), and thus require no shielding. For electrical systems, the weight of the shielding can be more than the weight of the sensor itself.

Fibers embedded in a composite material affect the physical integrity of the material. Claus *et al.* [26] examine the action of fiber optic sensors in a composite. In addition to discussing sensing capabilities of fiber optics, they demonstrate how an embedded fiber changes the stress patterns in the composite. They also discuss fabrication techniques that lessen the fiber's effect on the physical integrity. Their research concludes that the effects of the embedded fibers can be managed, but not eliminated. They also state that the long term affects of embedded fibers are not fully understood.

DasGupta, *et al.* [27], discuss the effects of fiber optic sensors on a composite at a micromechanical level. Their research identifies the same issues discussed in [26] and demonstrates work that may lead to both a short-term and long-term understanding of the effects of an embedded fiber on the physical integrity of a structure.

It is also difficult to connect fibers. Embedded fibers require output and pass-through leads at the joints of a composite structure. The stress concentrations at the boundaries differ from the continuous strain fields that exist in the interior of the composite, and these stresses can damage the fiber sensor leads. The development of fiber optic connectors to cross material boundaries remains as an important problem in fiber optic sensor research [28].

The cost of fiber optic sensors is typically dominated by the cost of the optical source that interrogates the fiber and the cost of the detector circuitry. The costs associated with the actual fiber are often ignored. In smart structures using embedded fibers, the fragility of fibers, and the reduction of physical integrity that they may cause, affects the sensor cost. It is apparent, however, that one way to limit cost and complexity is to limit the length of the embedded fibers and reduce the number of fiber optic connections.

#### 2.1.2.2. Piezoelectrics

Piezoelectric materials may also be used in smart structure applications [29-32]. Piezoelectrics are polarized dielectric materials that couple the effects of mechanical deformations to the flow of electric charge in the material. The polarization of piezoelectric materials, which can be controlled during their manufacture, determines the directional components of this coupling. A typical piezoelectric patch is shown in Figure 2.3. A conductive coating is placed on two opposing faces of the piezoelectric material. When the material is deformed, an electric charge appears on the two plates.

From this phenomenon, it is possible to construct both sensors and actuators by bonding piezoelectric patches to plates or beams in a structure [31]. The currents across the charged plates can be measured thereby determining the deformation in the material. This, in turn, can be used to measure deformations in the structure to which the piezoelectric is bonded.

The shape of the piezoelectric patches and the electrical charge distributions across their electrodes can be varied to perform spatial filtering. Polyvinylidene fluoride, PVDF or $PVF_2$, is a recently developed piezoelectric that can be manufactured as large sheets of thin films [29]. These films can be easily cut and shaped to match the contour of a structure or required filtering characteristics of a given application [32].

Both passive and active damping can be implemented using piezoelectric patches. In passive damping, the charge created as the patch deforms can be dissipated in a resistor connected between the conducting surfaces [30]. The energy of vibration is transferred to an electric charge in the piezoelectric material, which is then dissipated as heat in the resistor. In addition, the resistor can be replaced by an RLC circuit so that the damping mechanism only responds to certain frequencies of vibration. Passive damping requires no processing, but it is limited because it does not account for the effects that the local damping may have on other regions of the structure.

Active damping strategies can also be implemented using piezoelectric elements. Physical deformations can be induced in a piezoelectric material by placing an electric potential across the two conductive plates [33]. A controlled voltage source can be used to create specific structural deformations. A dual element approach can be used in which one set of piezoelectrics

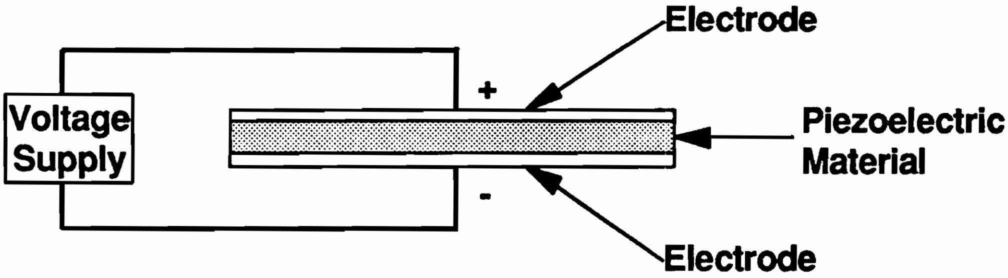


Figure 2.3. Piezoelectric patch

senses the deformations in a structure, and a second set of piezoelectrics is used for actuation. A processor receives signals from the sensing piezoelectrics, processes the signal, and outputs control signals to a second set of piezoelectrics to damp vibration.

#### 2.1.2.3. Proof-Mass Actuators

Proof-mass actuators consist of a proof mass and a surrounding body that controls the location of the mass along a predetermined path [34]. The actuator body includes magnetic materials that accelerate the proof mass, a shaft or track that defines the path of the proof mass, and a sensor that measures the location of the proof mass relative to the body of the actuator. The forces created by the actuator are applied to the structure to which the body of the actuator is attached. The magnitude of this force is determined by the weight of the proof mass, its rate of acceleration, and the mechanical coupling between the actuator body and the structure.

Zimmerman, *et al.* [35] discuss the operation of a proof-mass actuator controlled by an input voltage that creates the magnetic force that controls the proof mass. Once the coupling between the proof mass acceleration and the control algorithm have been identified, an analog to digital converter can be used to translate the digital signals created by the control algorithm into the required analog signals. Thus, the computational complexity is determined by the control algorithm, not by the actuator.

Proof-mass actuators have several significant limitations. The path length of the proof-mass is finite, so proof-mass actuators cannot create DC forces. Thus, they are not effective actuators for causing permanent deformations in a material. They can, however, be used to control vibration. The weight of an actuator affects the physical dynamics of the structure, so it is desirable to have a small proof mass. The weight of the proof-mass, however, is also a critical factor in determining how large of a force can be created by the actuator. Thus, in this respect it is desirable to have a large proof-mass. The proper weight of the proof mass is determined by balancing these factors.

#### 2.1.2.4. Shape-Memory Alloys

Shape memory alloys are materials that can alter their stiffness, stress, and strain distributions in a predictable and controllable manner [36]. This property enables them to alter their dynamic response to external stimuli. Nitinol is a shape memory alloy that is activated by heating. Nitinol exists in two different phases. In its low temperature phase, it is easily deformed. In its high temperature state, it returns to its original shape and becomes rigid. Changes in the chemical concentrations of nitinol can be used to vary the phase transition temperature between -50° C to 155° C [37]. Also, nitinol has a high enough resistivity that the heat created by an electrical current can cause this phase change.

Techniques have been developed for using shape memory alloys for vibration control in addition to inducing permanent structural deformations. Shape memory alloy fibers can be embedded in a structure. The number of embedded fibers and the geometry of their layout affects their mechanical coupling. This can produce spatial filtering affects similar to the ones discussed for piezoelectrics. The computational complexity required for shape memory alloy control depends on the specific characteristics of the implementation.

### 2.1.3. Control and Diagnosis Algorithms

Control and diagnosis algorithms must interpret sensor measurements and control algorithms must also determine what actions need to be taken by the actuators to control the structure. The possibility of large and widely distributed sensors and actuators creates conditions that are different from those encountered many other control applications.

For health monitoring applications, a diagnosis algorithm is needed to determine the current state of the structure. For vibration control, an algorithm is needed to control the actuators. Active damping techniques are used to target specific modes and frequencies of vibration, and these techniques are often needed to meet the performance goals of large space structures. Morgenthaler [38] shows that effective vibration control schemes should use a combination of passive and active damping techniques.

#### 2.1.3.1. Centralized Control and Diagnosis Strategies

Centralized control is based on the principle that full knowledge of the system's state is available at a central operating point. These methods model a system as matrix equations that relate system inputs to system outputs. The modeling effort requires that system states be defined that refer to physical parameters such as position, velocity, and strain. Inputs to the system, such as forces and electric signals, are then defined. The system must also have some way of identifying and controlling its current state. The control model determines how the current state information and the system inputs can be used to produce a desired output. Most algorithms use feedback measures that characterize how previous inputs have affected the system state. The effectiveness of the control system is determined by comparing the output to predefined performance goals.

Although perfectly linear systems do not exist in reality, many systems exhibit near linear behavior for a given range of interest and can thus be controlled using linear techniques [39]. A typical first-order mathematical model of a linear system is shown below.

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$$

$$\mathbf{y} = \mathbf{C}\mathbf{x}$$

where **x** is the current state vector, A is a matrix relating the current state to the expected change of state, B is a matrix relating the state change to the input, **u** is the input vector, **y** is the output vector, and C is a vector relating the current state to the output. The input vector is often separated into two vectors. The first vector is a disturbance vector that describes the affects of external events. The second vector describes the affects of events that can be manipulated by the system. From this system model, equations are derived that determine the required input to produce a desired output response. These equations generally require that a model is created that estimates the system state based on the sensor readings.

Effective control systems must be both observable and controllable. Observability refers to the fact that all of the system states in some way affect the output. Thus, an observable system state can be identified by examining the system output. In a smart structure, the system states are identified by the sensor measurements. The control model, however, must relate the measurement obtained by the sensors to the system state defined by the model.

Controllability refers to the ability of the system to exact some change on the system states. For a health monitoring operation, controllability is not required because the system only has to report on the health of the structure. For active systems such as vibration control or flight path control, controllability is essential.

Laub, *et al.* [40-42], have examined efficient methods for performing the large number of calculations that are often required in control problems. The algorithms they investigate are related in that their fundamental function is to solve a system of linear equations, and they are highly dependent upon the ability of the computer system to perform matrix multiplications. Laub comments that the second-order linear models that are often used to approximate the mechanical behavior of large space structures often contain sparse matrices of high order. It is also stated that these models often contain structure that enables them to be efficiently calculated using parallel techniques. They examine the execution of several control algorithms on multicomputer systems, such as the Intel iPSC hypercube, and find that a significant computational speedup is achieved using these parallel machines versus execution on high speed single processor machines.

Control techniques such as $H_{\infty}$ control and Linear Quadratic Gaussian estimators have been studied for vibration control [43,44]. These methods use different types of feedback loops, different algorithmic laws for measuring the current system state, and different assumptions concerning the system parameters in the control algorithm. The basic mathematical mechanics of these methods, however, remain fairly similar to the simplified linear system discussed above.

Although the basic form of the calculations for these techniques are often similar, the affected system parameters and the methods used for estimating the system state vary widely from one algorithm to the next. For instance, vibration control algorithms may be implemented in either the time or frequency domains. They may also use techniques that act on specific modes of vibration in the structure.

Several researchers have discussed algorithms to control vibration in a single beam [23,32]. Bailey and Hubbard discuss the use of piezoelectric materials for active vibration control of a cantilevered beam [32]. Their control algorithm is based on well known differential wave equations for a cantilever beam. Although their system demonstrates an effective use of active damping, the simplicity of the mathematical model of a single beam does not reflect the complex relationships that are present in a large structure.

Chiang and Fulton [45] examine the use of finite element models for control applications. These models attempt to partition the structure into individual areas. The system state and control commands are initially carried out for each of these areas individually. Boundary conditions are then imposed that modify these results to form a global solution.

Lee and Moon [46] discuss the use of modal control. Vibration in large structures occurs at specific orthogonal modes that are supported by the structure. Each mode can be defined by its modal shape, which determines the deformation it causes in the structure, and its frequency of vibration. In a discrete full order model, one actuator is needed for each mode in the system. In a reduced order system, an actuator is assigned to each dominant mode to damp out the majority of the energy of vibration. Reduced order systems must be designed so that the modes that are ignored by the control scheme do not create instabilities. A problem with modal techniques is that it is difficult to accurately identify the modes of vibration in large structures.

Although centralized control has been widely studied, there are several problems that need to be solved before it can be used in large smart structures. The computational requirements of centralized strategies grow rapidly as the number of sensors and actuators in the system increases, causing a centralized design to become intractable for large systems. Also, for large systems, it is difficult to accurately measure the dynamic coupling that occurs between distant regions. Small irregularities in the model can lead to instability [38], so errors made in modeling the coupling can cause failure.

#### 2.1.3.2. Decentralized Control and Diagnosis Strategies

Decentralized control algorithms partition a large system into several smaller and more easily managed subsystems. A critical element of successful decentralized control methods is the use of a system model that isolates the effects of stimuli and decouples the interrelationships defined in a control algorithm. Fundamental observability and stability problems impede development of decentralized control.

The control of the independent subsystems does not necessarily lead to global control [47]. Instabilities can arise from the coupling between the subsystems that are not accounted for in any of the localized control strategies. Passive damping can be used to decouple the effects of distant regions of the structure, and thus enable the use of a reduced order system model. Hierarchical systems that use localized control of subsystems and also allow for communication between the subsystems can potentially solve stability problems [48]. One goal of hierarchical control is to limit the information flow between processors [49].

Aldeen and Jamshidi report that interconnected large-scale dynamic systems typify the need for decentralized control [49]. They believe that traditional control techniques may fail because of both economic and technical considerations. Centralized algorithms in which the complete state of the structure is known at a central point are computationally complex. Also, the large number of data links required to transmit information to a central location are often economically prohibitive.

PACOSS is a study that examines *P*assive and *A*ctive *C*ontrol *O*f *S*pace *S*tructures. One example of decentralized control used in PACOSS is local direct velocity feedback [43]. Although PACOSS used a central controller, its design allows for decentralized approaches. In direct velocity feedback, each sensor is paired with an actuator [50]. Each sensor measures the acceleration of a specific location and its paired actuator attempts to damp out the motion at that point. In the PACOSS project, the actuator damping was limited so that it would not disturb the actions of other actuators in the system and thus maintain stability.

Lindner, *et al.*, [51] present a damage DLE algorithm for health monitoring in a planar truss. It is assumed that the truss incurs damage in amounts that will detract from the strength and function of the structure, but will not be so catastrophic as to cause it to completely fail. Also, only a small number, if any, of the truss elements are assumed to sustain damage during each iteration of the algorithm. Information concerning the global system behavior is required for damage detection, but the damage calculation for each element can be performed locally and is independent from the calculations for damage in other truss elements. Also, further work examines methods that use a reduced order model to detect regions of the truss that have sustained damage and then use a more detailed model to determine the exact element in the region that has been damaged. In this way, the algorithm begins to exploit the characteristics of physical locality of computation and hierarchical computation for health monitoring applications in smart structures. These characteristics are appealing for implementations on parallel computer architectures.

### 2.1.3.3. Neural Network Implementations

Artificial neural networks have been proposed for implementing the computations required in a smart structure. Neural networks have fundamental differences from the standard Von Neumann architecture that is the basis for virtually all commercial computers. Several different classes of neural networks have been examined, but they share fundamental similarities [52-54]. Neural networks consist of a densely interconnected set of simple processing elements, called neurons. Each neuron can perform only simple mathematical functions on its inputs to produce an output. The inputs to the neurons are system inputs or outputs of other neurons. The output of a neuron is usually based on a non-linear threshold function applied to a weighted sum of the input variables. Several neural network algorithms have been developed that can be used to determine the weights applied to the inputs and the appropriate threshold function for a given application.

One frequently used neural network algorithm is back propagation [53,54]. In this algorithm, the neural network is trained to produce a desired response. Training consists of applying sets of data to the neural network and specifying the proper output. The neural network then uses the difference between its output and the known output to adjust the weights of the neuron responses to make its answer closer to the predetermined response. At the end of the training sequence, the weights of the outputs are fixed and "real" data is applied to the neural network.

One desirable feature of back propagation is that it does not require an accurate model of the system. Instead, it depends on a training sequence that allows it to model the application and determine the proper response [55]. Lippman contends that neural networks are most effective in problems were high computation rates are required and where many different hypothesis can be examined in parallel [52].

Napolitano and Chen [56,57] examine back propagation based neural networks for smart structures. Specifically, they demonstrate the effectiveness of neural networks for state estimation of a vibrating cantilevered beam. They show that neural networks can be used to model both linear and non-linear behavior. They also indicate that neural network strategies are useful for implementing control strategies based on these state estimates. One drawback to their research, however, is that it only considers the action of a single beam. Connectivity problems are expected to arise when larger structures are considered. Thus, more research is needed to determine the performance of neural networks in large structures.

Artificial neural networks for smart structures have also been investigated by Grossman and Thursby [58-61]. Their research deals with several different smart structure applications, and it outlines common characteristics for computing systems for smart structures. Grossman states that methods for processing the large number of separate sensor signals that exist in smart structures need to be addressed [58]. He feels that the processing effort is computationally intensive and is fertile ground for massively parallel architectures.

Grossman views neural networks as having two main advantages for smart structures: they are able to analyze complex sensor signal patterns and they can generate the necessary control signals quickly [58]. Grossman discusses the simulation of a smart electromagnetic structure in which a control task that matches a transmitter frequency to a given received frequency must be performed in a time shorter than 15 gate delays [60].

Grossman also details work that has been performed for the development of optical processing for smart structures [58,59]. He sees optical processing as an effective solution for several reasons. Other aspects of smart structure systems are already implemented using optical techniques. Grossman's work cites the use of optical signals to energize actuators in addition to fiber optic sensing applications. He recognizes that one benefit of all-optical processing is that it eliminates the necessity of converting between optical signals and electrical signals. The time saved by eliminating these conversions could speed up the computations and enhance the system's real-time performance. The all-optical networks proposed by Grossman are still under development, but he feels that many of the processing algorithms proposed for all-optical networks can be demonstrated using electro-optical networks.

### 2.1.4. Processing and Data Requirements

It is necessary to understand the characteristics of an intended application before an appropriate computer architecture can be chosen. This section discusses processing and data communication requirements of smart structures.

#### 2.1.4.1. Processing Needs

Smart structures require data processing to translate the data received from the sensors into actuator control signals. This translation takes place in several stages. First, raw sensor signals are converted into measurements that are compatible with the control model. Next, the sensor measurements are used to estimate the state of the structure. For health monitoring applications, the description of the structure's state is the desired output. One additional level of processing may be required to relate the state to information that can be interpreted by a human operator. For vibration control algorithms, the actuator control signals are calculated from the relationship between the current state and the desired state. One final level of processing may be required to convert the control system's outputs to signals that can drive the actuators.

### 2.1.4.2. Communication Needs

The specific data communication requirements depend on the control or diagnosis algorithm used. However, several basic characteristics exist for all of the feasible implementations. Because sensors are distributed throughout the system, data communication is needed to move information from the sensors to the processor or processors. For centralized processing, raw sensor measurements multiple points in the structure must be delivered to a single location. The number of data paths required in the structure may be reduced by multiplexing several sensor measurements onto a single communication link [62]. For distributed architectures, shorter data links can be used to transmit sensor readings to a nearby processor. Additional communication links are then used to transmit the processed sensor data or other information to neighboring processors. Once the desired action for the control system has been determined, the system output is generated and communicated throughout the system.

For centralized approaches, the system output originates from a central location. For a health monitoring application, this output could be transmitted via an external data port. For vibration control, the output consists of actuator commands that must be distributed to each of the actuators in the system.

For distributed approaches, the system output consists of a model of the current state and control information that local processors send to nearby actuators. For health monitoring applications, additional processing must be performed on the distributed output to create a report of the overall system condition that can be transmitted to an external port. For vibration control, each actuator receives control signals from a local processor.

### 2.1.4.3. Real-Time Processing Constraints

In real-time computing, results are correct only if the proper logical result is achieved and the time at which the result is available meets predefined timing constraints. The constraints usually refer to an interval of time during which the result is expected. For some systems, the timing constraints are represented as hard deadlines that define a specific time before which the results must be achieved. For hard deadlines, it does not matter how early a result arrives so long as it does not arrive after the deadline. Williams claims that a system may sometimes have to be monitored for weeks at a time to determine the worst case timing for the system [63]. The probability that a hard deadline can be met can sometimes be determined by knowing the average time at which a result is expected and the variance of this time.

Stankovic outlines the characteristics of effective real-time processing [64]. A popular misconception that Stankovic discredits is that real-time computing is the same as fast computing. He states that although speed can help a computer system meet timing constraints, it is not synonymous with real-time operation. Instead, predictability is the most important characteristic of real-time systems.

The examples presented in Section 2.1.1 show the diversity of capabilities that may be required of a smart structure. In some of the applications, such as periodic health monitoring, a relatively large amount of time is available for computation, and thus the real-time constraints can be relaxed. For applications such as vibration control, however, the timing of computation and communication is a critical constraint. Stankovic discusses scheduling theory for real-time systems that attempts to minimize the resources that are needed by a computer system to fulfill real-time constraints [64].

#### 2.1.4.4. Physical Design Constraints

Requirements for computers for smart structures are different than many other applications because of physical design constraints. Many smart structure applications involve aerospace applications where weight must be minimized. Also, the processors and data communication links must be attached to, or embedded in, the structure. Due to the mechanical concerns created by embedded components, the designer of a smart structure must try to limit the number and size of processors and communication links.

The location of sensors and actuators is dictated by the size and shape of the structure as well as by the desired performance. Thus, the placement of the inputs and outputs, as well as the location of processors and data links that are connected to these data resources, is determined by the application. The paths for data links must consider not only the location of the processors, inputs, and outputs being connected, but also the location of the support beams or other elements in the structure that can carry the links.

### 2.1.5. Previous Research in Computing For Smart Structures

Most research in smart structures has merely stated that processing will be required without looking at how this requirement might be met. Islam and Ammar discuss computing in real-time distributed systems [65]. Their system model is a set of sensors and actuators that attempt to measure and control unspecified environmental variables. Although their model also examines applications, such as radar systems, that are not related to smart structures, many of their results are relevant to smart structures. They state that the majority of a distributed real-time work load is filled by a set of tasks that are repeatedly executed at regular intervals. They also state that fault tolerance is needed in large distributed systems and algorithms must be designed so that the processing loads can be redistributed as system failures occur.

Ahrens and Claus [66] recognize the importance of the processing architecture of a smart structure system. They investigate a real-time controller designed to execute vibration suppression algorithms for smart structures. Their system contains a single processor which is used to control vibration on a cantilever beam. The processor uses a reduced instruction set computer (RISC) architecture that provides high average and burst computation rates. The controller also contains local memory, a math coprocessor, and digital-to-analog and analog-to-digital converters that provide control signals to external sensors and actuators. In addition to computing performance, their design considers power requirements, weight constraints, fault tolerance, and the importance of real-time design constraints. For larger systems, more powerful processors must be developed to account for the physical complexity of large structures.

Chiang and Fulton [67] discuss the use of multicomputers for solving finite element equations that govern the non-linear dynamic structural response of a simply supported arch. They test their data on the Intel iPSC and the Flex/32. The iPSC is a multicomputer that has a hypercube topology and uses a packet-switched communication protocol. The Flex/32 is a shared memory computer that uses a 20.5 megabits per second (Mbps) communication bus to connect the processors and memory. They relate their arch example to automobile crash test evaluations, and thus their analysis is related to health monitoring applications that may be used in smart structures.

Chiang and Fulton discuss two different algorithmic approaches. "Physically parallel" approaches assign each processor to a specific region of the structure. The calculations converge to a solution in an iterative manner as regional solutions are compared and boundary condition modifications are made. "Algorithmically parallel" approaches assign each processor to a specific set of calculations that arise from a mathematical model of the entire structure.

Chiang and Fulton demonstrate that multicomputers can be effectively used to solve structural dynamics problems. They show that as long as the ratio of the communication time to the computation of the processors was small, a nearly linear speedup was achieved as additional processors were added to the system. Thus, they demonstrated that parallel techniques can be effectively applied to dynamic modeling problems. They also report that the shared memory system was more efficient than the distributed memory architecture for both the physically parallel and algorithmically parallel implementations for small problems, such as a simply supported arch. Their research does not, however, examine which architecture is more effective as the problem size is increased.

As has been noted, little research has examined the computational requirements of large smart structures. Thus, processing demands are ignored as the complexity of proposed smart structure applications expand and the need for processing grows. The next section describes a computer architecture that may fill these spiraling needs.

## *2.2. Multicomputer Systems*

The need for processing is a characteristic of large smart structures that transcends all applications. Some applications may require thousands of sensors [19]. This large number of sensors requires a significant amount of computation and communication even for low sampling rates. Multicomputers may be able to meet this demand in a cost-effective manner.

### 2.2.1. Definition of a Multicomputer

Multicomputers, also known as distributed memory computers, are a type of parallel processor consisting of multiple processing nodes connected by a communication network. Each processor, also referred to as a node or a processing element, has computing, memory, and communication resources. The computing resources perform the processing assigned to the node. The memory stores both program and data. The communication facilities are used to transmit and receive data from other processing nodes.

The performance of a multicomputer can be judged on many different criteria, including aggregate processing power, communication throughput, and message passing latency. Each of these measures depends on the application being executed, the manner in which this application is mapped to the nodes of the multicomputer, and the topology of the multicomputer's interconnection network. The following sections discuss multicomputers and their performance measures.

### 2.2.2. Processors

The processing nodes in multicomputers are typically similar to those of simple single processor machines. The Ncube/10 uses multiple custom-designed 32-bit processors operating at 7 MHz [68]. The Ametek 2010 multicomputer uses 25 MHz Motorola 68020 processors [68]. Intel's commercial multicomputers include the iPSC/2, introduced in 1987, and the iPSC/860, introduced in 1990 [69]. The processing nodes of these multicomputers use 16 MHz 80386 and 40 MHz i860 processors, respectively [68,69]. Intel's latest multicomputer, the Paragon XP/S, uses several i860 processors for computation in each node [70]. Other recently developed massively parallel computers include Thinking Machines Corporation's CM-5 that uses a Sun SPARC processor at each node and the Ncube 2S that uses a custom-built 64-bit processor [70]. The amount of memory in these machines varies from 128 kilobytes to 64 megabytes per node. The processors in multicomputer nodes may implement features such as memory caching, instruction pre-fetching, floating point operations, timer operations, and DMA transfers [71]. As can be seen by these examples, the power of multicomputer nodes has steadily increased over time as the power of commercial microprocessors increased.

The examples above show that the processors and memory of a multicomputer's processing nodes are essentially the same as in single processor machines. The support for interprocessor communication is what distinguishes multicomputer nodes from single processor computers. Since message passing is used to share data between the nodes, communication becomes a major function of the node. Early multicomputers used store-and-forward message passing and required processing to be performed on messages passing through the node. More recent multicomputers use specialized routing hardware that guides messages through a node without affecting the performance of the main processor in the node [72].

In addition, features are often needed to deal with the specialized nature of the algorithms that are commonly executed on multicomputers. Programming languages have been modified to allow for concurrent applications. Also, multicomputers nodes sometimes contain coprocessors to support vector processing applications.

Transputers are a type of multiprocessor that are specially designed for parallel processing. A transputer is a self-contained computer on a single chip. A transputer contains a processor, local memory, and communication links so that it can interface with other transputers [73]. Computer architectures are sometimes categorized on the basis of how many different commands the processor can execute. Reduced Instruction Set Computers (RISC), as opposed to Complex Instruction Set Computers (CISC), use a small number of commands that require only a single instruction cycle to execute. Transputers use an instruction set similar to RISC architectures, and they use fewer registers than are typically found in commercial microprocessors. They do, however, use high clock speeds and the processing power of transputers has kept pace with microprocessors [73,74]. Newer transputers contain more advanced features such as memory caching, pipelined instruction fetching, and floating point arithmetic [74]. Also, optional chips are now available that move support functions such as message routing to hardware.

The primary benefit of transputers is that added processing nodes are accompanied by their own communication links. Transputers typically have four independent full-duplex communication channels on each chip, and the data rates of the channels maintain a balance between the communication and computation resources [74].

The proper multicomputer node architecture for a given application can be chosen by examining several factors. First, the necessary computational power of the node's primary processor must be determined. Additional functionality can be added by increasing the capability of the processor or by incorporating specialized modules into the node. These modules include floating point coprocessors, memory management units, and routing hardware. Factors such as whether to use a RISC of CISC architecture also help to determine the processor's performance for an intended application.

### 2.2.3. Interconnection Networks

Multicomputer nodes contain no shared memory, so data must be transferred between nodes over the interconnection network. Thus, the overall performance is heavily dependent on the interconnection network. The topology of an interconnection network defines the number and placement of the communication links that join the processing nodes of a multicomputer. The following sections discuss ways to measure the performance and capabilities of several network topologies that have been proposed.

#### 2.2.3.1. Performance Issues

The performance of multicomputers depends on parameters, such as the speed of the processors and the bandwidth of the communication links, that can be expected to progress with fundamental technologies. The organization of these building blocks also affects the performance of the system. The performance measures discussed below depend largely on the organization of the system and are somewhat independent from the technological state of individual components.

The performance of a computer architecture depends on the application being executed. Multicomputers must balance the amount of processing and communication in the system. One key factor that determines the optimal topology of the interconnection is the way in which the application is partitioned into individual processing tasks. The granularity, or grain size, of the partition defines the size of each processing task. The processor assignment determines which processors in the multicomputer are used to perform specific processing tasks. The partition, granularity, and processor assignments for an application influence the overall multicomputer performance.

One performance measure is message latency, the amount of time it takes a message to travel between two processors [75]. Message latency determines how fast processors can receive new data and how quickly the results of one processing node can affect the results of other processing nodes. Message latency depends on how long it takes to prepare the message for transmission, how far the message must travel, how busy communication links are, and how long it takes the receiver to process the message. The dimensions of multicomputers are typically very small, therefore the physical distance between processors is not the determining factor of communication path length.

Early multicomputers used store-and-forward techniques to transmit messages [68]. Store-and-forward routing allows communication between two distant nodes by passing messages to a series of intermediate nodes. Each intermediate node receives the messages in a buffer, examines the message's header information to determine whether it should keep the message or transmit it on another communication link, and then waits until that communication link is idle to transmit the message. The time it takes to receive and examine the header and the time the message has to wait before the proper communication link is available are the dominant factors in the transmission time of a message. Thus, for store-and-forward techniques, distance is measured by how many separate nodes the message must pass through before reaching its destination. The average internode distance is an estimate of the typical performance of the network [76]. The maximum internode distance, also referred to as the network diameter, is an estimate of the worst-case performance [77]. These measures, however, assume that loads on the links and at nodes are uniform. Message locality indicates the average distance that messages generated in a particular application have to travel.

Another measure, throughput, indicates how much data can be transferred by the interconnection network in a given period of time. It is important to relate system throughput to latency by examining how the network reacts to loading and how gracefully performance declines as the communication links become congested. As traffic in the network increases, both throughput and latency tend to increase. The network loading, in turn, is affected by message locality. For example, if a message is transmitted between neighboring processors, it travels over only one communication link. If this same message is transmitted between distant processors, it travels over several links, thus generating additional traffic. Throughput for a given degree of latency can be increased by reducing the distance between the nodes [75].

Another important characteristic of multicomputer networks is the routing mechanism that guides messages from source to destination [78]. Agrawal [76] states that topologies should have a small number of communication links and easy routing rules. Thus, a processing node must be able the determine the path that the transferred data will take as it travels from source to destination without incurring excessive processor overhead. Each message begins with a header that provides information such as the length of the message, the source of the message, and the destination of the message. Some routing schemes set up the entire route that the message will take before it is transmitted, while other methods allow the intermediate nodes that receive the message to decide where to send it.

Processing nodes can use lookup tables, also known as routing tables, to determine which link should carry an outgoing messages [77]. These tables contain one set of paired elements for each possible message destination. The first element of the pair is a possible message destination. The second element of the pair is the next link that should be used to send the message toward its destination. The simplest algorithms are static algorithms in which the entries in the routing table are determined before the application is run. Adaptive routing techniques can also be employed that monitor traffic loads at links and update routing table entries so that messages are guided along lightly loaded links. Adaptive techniques require additional messages to be sent between processors so that processors can learn about the current traffic levels in the network. Static techniques are less complex and are easier to implement. Adaptive techniques, however, offer the potential of being able to maximize the communication resources of the network.

Wormhole routing is a method that is commonly implemented in recent multicomputers to establish the transmission path between the source and destination of the message [68,79]. In this routing strategy, each message is broken up into individual packets. The first packet of the message contains status information such as the source, destination, and length of the message. A node receiving this first packet examines it to see whether the message has reached its destination or has to be transmitted on another link. If it needs to be transmitted on another link, the node waits for that link to become idle and then begins transmission. If the packet is at its destination, it is received by the node. Additional packets of a message are transmitted each time a previous packet is forwarded through the network. If a previous packet cannot be forwarded because some link in the message path is in use, then the previous links in the message paths are held even though no packets are transmitted on them. Using this method, the message "worms" its way from its source to its destination.

Some characteristics of wormhole routing are worthy of note. Only a few packets of a message need to be stored at a given node during transmission. This limits the amount of buffer space required at each node. Also, once the message has reached its destination, the links along the message path are dedicated to that message and thus the message transmission can be completed at the maximum transmission rate of the links. In this way, wormhole routing requires an initial delay to create the message path, but then swiftly completes the transmission. All packets in the message follow the same path and are received in the order that they are transmitted so the receiving node can easily reconstruct the initial message from the received packets.

A potential problem of wormhole routing is that a blocked message may hold a series of links for a long time as it waits for transmission. To alleviate this problem, Dally [80] has proposed virtual-channel flow control in which each link is split up into a number of independent channels. This limits the bandwidth of each channel between any pair of processing nodes, but it has been shown that this drop in peak performance is made up for by reducing the overall congestion encountered in the network and increasing the network throughput.

A final characteristic of wormhole routing is the affect of increased traffic on message latency. Several researchers have shown that message latency in multicomputers that use wormhole routing rises slowly as the amount of traffic in the network increases until the network approaches saturation [81,82]. At this point, small increases in traffic cause large increases in the message latency.

Another important measure of multicomputer networks is their fault tolerance [76]. Different network topologies have different degrees of fault tolerance. One measure of fault tolerance is the number of independent paths that exist between any two nodes in the network. Fault tolerance is also measured by how many links or nodes must be removed before the network becomes disconnected. A disconnected network is one with no communication paths between some pair of nodes. Other studies of fault tolerance examine how a processor failure affects the workload of surrounding processors.

In any performance comparison, it is important to compare performance gains against cost. Cost can defined in terms of a compromise between two performance measures or in terms of system characteristics that can be translated into dollars once the exact specifications of the system are determined. One cost measure of a multicomputer interconnection network is the number of links in the network [76,83]. The number of links required per node is also a cost factor.

The cost of expanding the network, also known as the scalability of the network, is important for some applications [76]. This cost can be considered by examining how the existing network must be changed to accommodate additional processing nodes. For some topologies, only a small number of communication links have to be added when additional nodes are incorporated into the design. Other topologies, however, require that the structure of the communication controllers in all of the existing processing nodes be changed when additional nodes are added. The expansion criteria should also examine how traffic in the existing network links is be affected by adding nodes to the network.

#### 2.2.3.2. Classifications

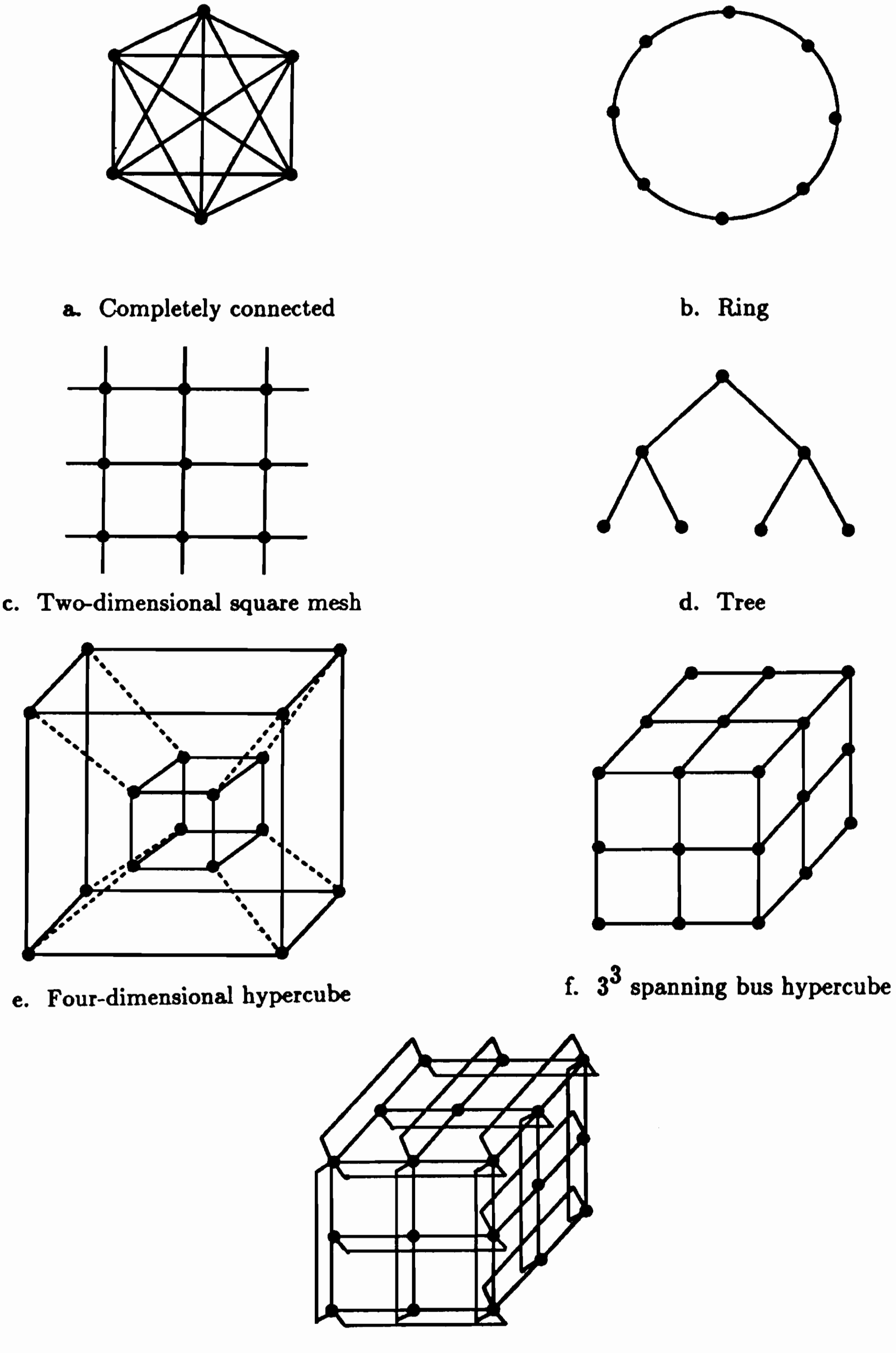
Multicomputer networks can be classified according to several different criteria. One such classification is based on the number of stages that exist in the communication links between the nodes [78]. In single-stage networks, also known as direct networks, each communication link is connected only to processing nodes. Processing nodes that do not have a direct link between them must pass data through intermediate nodes when they communicate, but the individual links in this communication path are directly connected to pairs of processing nodes. Multistage networks are created by the interconnection of one or more stages of switches. The inputs to the first stage of switches are attached to the output of the processing nodes and the outputs of the last stage are connected to the inputs of the processing nodes. The switches in the intermediate stages of the network are attached to switches in their neighboring stages. By changing the state of the switches, different connections can be created.

The ability of multistage networks, such as the multistage cube and the augmented data manipulator network, to provide communication for parallel processing elements has been examined [84,85]. These networks, however, possess several qualities that make them undesirable in large systems such as large space structures [86,87]. Multistage networks tend to require the same communication path length between any pair of processors, and thus cannot take advantage of localized processing and data sharing. Also, the internal stages of the multistage networks will add to the number of communication links and components that are embedded in the structure. Thus, this research examines only single-stage interconnection networks.

A single-stage interconnection network uses either point-to-point links or bus links. Point-to-point links provide a direct connection between a pair of processors. They are well suited for fiber optic implementations and can employ simple protocols since only two nodes attempt to transmit on the link. Bus links allow more than two processing nodes to be attached to a single communication link. Only one of the nodes is allowed to transmit at any given time, but this transmission is received by all of the other nodes that are connected to the link. A bus contention protocol is needed to decide which node has permission to transmit. Topologies based on each type of communication link are discussed below. This discussion initially introduces point-to-point versions of the given topologies and then describes bus topologies that are derived from the point-to-point networks.

#### 2.2.3.3. Specific Networks

The topology with the greatest and least connectivity are the completely connected network, shown in Figure 2.4a, and ring network, shown in Figure 2.4b, respectively [83]. The completely connected network is by its very nature a point-to-point network. The ring network, on the other hand can be implemented by using a point-to-point link between the neighboring processing nodes in the ring or it can be implemented as a set of nodes connected to a single communication bus. Although the completely connected graph directly connects each pair of



a. Completely connected

b. Ring

c. Two-dimensional square mesh

d. Tree

e. Four-dimensional hypercube

f. $3^3$ spanning bus hypercube

g. Torus

Figure 2.4. Interconnection network topologies

processors, and thus provides the shortest paths for communications and a high level of fault tolerance, the cost of such networks is prohibitive for large systems [76]. The ring network, on the other hand, uses the fewest number of communication links possible for a connected set of processing nodes and can easily be expanded. For ring networks, the average number of nodes that a message passes through from the its source to destination grows linearly with the size of the network. Also, each link must pass messages for several different processors. Thus ring networks become saturated as the amount of communication increases. There are only two different paths between any pair of nodes in a ring network, and thus it has a low tolerance for either node or link failures. The study of communication topologies deals with the compromises that must be made between the low internode distance between processors in completely connected networks and the low cost of ring networks.

In a $D$-dimension mesh network, processing nodes are placed in a lattice arrangement, and each processor uses a separate communication link to communicate with each of its $2D$ neighboring processors. This interconnection represents a compromise between a completely connected network and a ring network. A two-dimensional or square mesh network is shown in Figure 2.4c. For a square mesh network, the network diameter and the average number of links that a message must traverse from source to destination increases proportionally to the square root of the number of nodes in the system [83]. The fault tolerance of mesh networks increases as the dimension of the network increases. Also, expansion of the mesh affects only the processing nodes that are on the outer edges of the network. Mesh networks are usually constructed using point-to-point links between each of the nodes, although it is also possible to use a set of buses in which each collinear set of point-to-point links is replaced by a single communication bus. Also, variations of mesh networks can be implemented that attempt to conform to the traffic patterns that are expected in algorithms executed by the network.

A tree network is shown in Figure 2.4d [76]. Trees are constructed by beginning with a single processing node then adding levels of nodes. Each node in an added level is only connected to one node of the previous level. Tree networks require only *N-1* communication links for an *N* node system, and they are suitable for implementing hierarchical algorithms. The network diameter of tree networks is linearly proportional to the number of levels in the structure, so the diameter is proportional to the logarithm of the number of nodes.

In this form, tree networks demonstrate poor fault tolerance since the failure of any node disconnects all nodes below the failed node from the rest of the network. Also, the communication links at the top of the tree usually tend to become saturated. Several variations of tree topologies have been proposed that include additional links between processing nodes at the same level to solve these problems [75,76]. The added links increase fault tolerance and tend to distribute the communication traffic more evenly.

The binary hypercube, shown in four dimensions in Figure 2.4e, is a popular topology for commercial general-purpose multicomputers. This is because hypercubes exhibit strong connectivity, regularity, and symmetry [87]. The connectivity enhances fault tolerance and the regularity and symmetry allow for relatively simple routing algorithms. The connectivity of the network is determined by assigning a distinct *D*-element binary vector to each of the $2^D$ processors in the network. Every pair of nodes that differ in only one of *D* vector coordinates are joined by a point-to-point communication link. Thus, each node requires one communication link for each dimension of the network. For this topology, the average internode distance and the network diameter increase proportionally to the base two logarithm of the number of nodes in the network [76]. Complete binary hypercubes only support a number of nodes that is an integral power of two, and expanding the network to a larger dimension requires that an additional communication link be added to each of the existing processors in the system. Thus, binary hypercubes are not easily expanded.

Many variations of the binary hypercube have been examined [83]. The $W^D$ spanning bus hypercube is a $D$-dimensional hypercube that is similar to the binary hypercube, but instead of having a pair of processing nodes connected by a single point-to-point link in each dimension, a set of $W$ processors connected to a communication bus is used. Thus, each bus in the network connects $W$ processors and each processor in the network is connected to $D$ different buses. The configuration for a $3^3$ spanning bus hypercube is shown in Figure 2.4f. Messages traveling through a spanning bus hypercube require at most $D$ transmissions to travel from source to destination, but each transmission is over a bus shared by $W$ processors. Thus, a fault in one of the network's links affects several processors. It is easier to add nodes to the spanning bus hypercube than to the binary hypercube because the value of $W$ can be increased without affecting the structure of the existing nodes.

The torus topology, shown in Figure 2.4g, is similar to a spanning bus hypercube except that the busses that join a set of $W$ nodes is replaced by a ring of $W$ point-to-point links. Both the network diameter and the average number of hops required to transmit messages between nodes is larger in the torus than in the spanning bus hypercube. As with the spanning bus hypercube, additional processors can be added to the torus by increasing the value of $W$. The abundance of point-to-point links between neighboring nodes increases the fault tolerance of the network and is suitable for algorithms with localized communication. The increased number of links in the network creates a large number of independent paths between any two nodes in the network that can increase the complexity of routing algorithms.

The interconnection networks discussed above are general topologies that are suitable for a variety of applications, but they rarely match the exact traffic patterns of the algorithm. Application specific interconnection networks have been designed by first examining the communication patterns that exist and then creating a network topology that supports these patterns. This scheme has been used to create architectures that support the regular communication patterns exhibited by matrix algorithms [88]. A drawback to these custom topologies is that a new topology has to be created for each new application. Other architectures have been proposed that use generalized interconnection networks as low-level building blocks of the network and then use application specific high-level topologies to connect the low-level building blocks in custom designed patterns [87]. Reconfigurable architectures that allow the user to adjust the interconnection network topology to fit a specific application have also been proposed [89]. A single reconfigurable architecture can be used to implement several different application-specific topologies.

### 2.2.4. Previously Investigated Application Domains

Multicomputers have been used in several different classes of applications. The research performed in these areas is discussed in the following sections. This discussion concentrates on the characteristics of the applications that make them suitable for multicomputers and outlines initial results.

#### 2.2.4.1. Neural Network Implementations on Multicomputers

The use of neural networks does not exclude the use of multicomputers. In fact, it is possible to implement some neural networks on multicomputers in an efficient manner. Research by Ghosh and Hwang [86] has shown that neural network computations are often distributed among clusters of neurons. They show that even though rich connectivity exists between the neurons in a neural network, only a small subset of the connections are used consistently. Thus, if all of the computations of a cluster are performed by a single, more powerful processor, a group of processing nodes can be created that perform most of their calculations on data in their own memory, and only occasionally need to transmit to other nodes. This configuration is essentially the same as a multicomputer. Ghosh and Hwang contend that since interprocessor communication is often a bottleneck in a highly concurrent system, limiting the number of processors can improve performance.

The value of collecting multiple neurons into one larger processor depends on the specific application. By reducing the number of processors, the number of communication links can also be reduced. The speed of the system, however, may be adversely affected. In many cases, a large single processor may not be as cost effective as several small processors.

Ghosh and Hwang [86] attempt to characterize neural networks to determine a suitable degree of parallelism. From this characterization, it is possible to determine whether a specific neural network approach can be effectively implemented on multicomputers. They decompose the neural network's neurons into regions. Initially, they partition the neurons into sets of core groups with approximately the same number of neurons in each group. Core groups are collections of neurons with a relatively high degree of connectivity to each other. Next, the influence region of each core group is defined to be a set of core groups that have a large number of connections to the neurons of the core group for which the influence region is being defined. All core groups not in the influence region of a given core group are said to be in a remote region of the core group. Once these three regions are defined for each of the neurons, it is possible to determine an efficient mapping of the neurons onto nodes of a multicomputer. The relation of the influence and remote regions of a given core region helps to define the topology that may be suitable for the application.

Ghosh and Hwang also discuss details of large neural network implementations on multicomputer networks [90]. First, they note that the computation requirements of fully connected neural networks increase quadratically with respect to the number of neurons, and thus prohibit the use of fully connected systems for large networks. For this reason, they feel that large systems must imitate both analytical and neurological examples that exhibit hierarchy. Ghosh and Hwang also recommend asynchronous packet switching with distributed control because of the small packet sizes that they envision in the network and the large number of virtual neighbors that exist in such systems [90]. They also recommend that the networks use point-to-point links. They feel that multistage interconnection networks are too large to provide fast communication and also do not benefit from locality that is exhibited in a neural network implementation. They feel that a hybrid system that uses both point-to-point and bus links may be effective for medium sized systems that frequently require broadcast transmissions. Although Ghosh and Hwang propose hypernets as an attractive point-to-point architecture for these systems, they also recognize the benefits of meshes, hypercubes, and binary trees.

Zhang, *et al.* [91] examine implementations of a back propagation neural network algorithm on both two-dimensional mesh and hypercube multicomputers. They find that modifications to neural network algorithms are required to achieve maximum performance given the architectural features of multicomputers. Their work shows that the smaller network diameter of the hypercube interconnection network allows it to perform better than two-dimensional grid networks.

Steck, *et al.* [92] use an Intel iPSC/2 multicomputer to implement neural networks. Although they address many of the issues mentioned above, they feel that multicomputer implementations of neural networks are needed only because large neural networks are not yet realizable. Their research does demonstrate, however, that neural networks can be effectively implemented on multicomputers.

#### 2.2.4.2. Real-Time Processing with Multicomputers

The design and operation of a multicomputer architecture affect parameters that are critical for real-time processing. The author [93] examines the affects of communication protocols and network topologies on real-time performance. He shows that protocol changes may affect the average message delay differently than they affect the variance of the message delay. Thus, multicomputer designs with a small average message latency are not necessarily suitable for real-time applications.

Williams [63] discusses the characteristics that real-time multiprocessors should possess. He mentions the need for software that can take advantage of the application's features and the inherent abilities of the interconnection network. He states that flexibility can help overall performance. This flexibility includes the ability to transfer computational tasks between processors to evenly distribute the system workload. He states that message passing is necessary to provide this flexibility. He also emphasizes the importance of the relationship between the interconnection network and the ability to efficiently distribute tasks. To limit traffic, he believes that the interprocessor communication mechanism should not introduce background communication. He also states that the current trend for multicomputer systems is to eliminate shared memory from the system and to use point-to-point links. Another need that Williams discusses is the need for an extended monitoring period to determine the worst case timing of a system.

Transparency is a term referring to the access that the user has to the internal workings of the computer. In transparent systems, the user does not know what processors, memory units, or other devices are being utilized when the computer executes the users commands. Fiddler, *et al.* [94] discuss the tradeoff between transparency and real-time performance in multiprocessor computer systems. They indicate that transparency can be increased in multicomputers by increasing the complexity of the software. The additional functionality provided by the software, however, can degrade the performance of the system. Thus, the performance of a real-time system is inherently coupled to the degree of transparency in the system.

Shin [95] developed a distributed computer architecture to examine real-time issues. He believes that distributed systems with point-to-point links are ideal for real-time applications because of their high degree of reliability and their potential processing power. Two key elements of a real-time distributed network that are discussed by Shin are the affects of message buffering and routing methods. Store-and-forward routing techniques are deemed to be too unpredictable for real-time computation and thus wormhole routing is recommended. He states that maximum message passing delay analysis is more important than average delay analysis for real-time systems. Shin also examines methods for task scheduling. One difficult aspect of task scheduling is that there is no global clock for a distributed network. Software techniques are discussed in which each processor transmits the time of its local clock at regularly scheduled intervals. In this way, processors can compare their local time to the time held by its surrounding processors in an effort to add some degree of synchronization to the network.

#### 2.2.4.3. Signal Processing on Multicomputers

Signal processing is an application domain that demonstrates a high degree of parallelism, and thus can benefit from the processing capability of multicomputers. Control signals are often determined by matrix calculations which, due to their regular structure, are suitable applications for multicomputers. Matrix operations such as gaussian elimination and matrix factorization can be effectively mapped onto multicomputers [96,97].

Multicomputers can also be applied to image processing applications. Chandran and Davis [98] examine the use of distributed memory and shared memory parallel architectures for detecting features in a visual image. They discuss the computational intensity of image processing algorithms and examine how to exploit the spatial parallelism of image processing techniques. They note that research is needed to determine the optimal partition size for an application. This size represents a tradeoff between the processing power created by adding processing nodes to the system and the additional communication that is required to support these added nodes.

Daniel and Teague [99] discuss the use of multicomputers for filtering images to clarify objects in the image. For this application, neural networks are mapped onto the processing nodes of a multicomputer. As with the previous algorithm, multicomputers can provide the processing power that is required for this computationally intensive problem.

Fast Fourier Transforms (FFTs) are another type of signal processing algorithm that has been mapped onto multicomputers. Zhu [100] discusses the performance of an FFT algorithm mapped onto a hypercube multicomputer. He shows that the regular structure of the FFT maps well onto hypercube architectures. For the proper problem sizes, the processing power of the nodes can be successfully utilized without being hampered by communication overhead.

## *2.3. Application of Multicomputers to Smart Structures*

The first section of this chapter examined the processing and data communication needs of smart structures. The second section described the features and capabilities of multicomputers. This section integrates these concepts by describing the use of multicomputers in smart structures.

### 2.3.1. Potential Advantages of Multicomputers

Several researchers have addressed how to perform distributed control tasks using embedded processors. Mazur, *et al.* [7] and Wada, *et al.* [101] discuss the feasibility of a hierarchical computer architecture for smart structures. Talat [2,3] briefly discusses requirements for a computer system for smart skin applications. These researchers mention the possibility of having many small processors throughout the structure that feed information to a central processor. The distributed processing capabilities of multicomputers naturally complement the distributed sensors and actuators that make up a smart structure. Thus, multicomputers can potentially implement hierarchical processing schemes for smart structures in a cost-effective manner.

The processing of sensor measurements is necessary in all smart structures. Smith [62] cites the potential advantages of placing processing nodes near groups of sensors, and in this way integrating the sensor and processing functions. He feels that this integration allows designers to scale the sensor system to the size of the structure. Also, the average length of the feedthrough fibers that send sensor measurements to processors can be shortened when distributed processors are used. Haskard [102] discusses the use of microprocessors with silicon-based sensors for signal conditioning. He indicates that these microprocessors can perform system level functions in addition to executing diagnostic routines and calibrating the sensor.

Herwaarden and Wolffenbuttel [103] discuss the design of sensors that can be easily interfaced to microprocessors. They emphasize silicon sensors, but several ideas are applicable to all types of sensors. They discuss the use of sensor busses in which processors are placed near the sensor locations and the sensor measurements are multiplexed over a bus to surrounding processors, thus improving the flexibility of sensor networks in large systems.

The number of sensors and actuators in large smart structures will require large amounts of computation. Since the processing power of multicomputers can be scaled with system size, multicomputers can potentially satisfy these processing requirements. Also, a multicomputer interconnection network can potentially reduce the number and length of the embedded communication links that distribute sensor information and actuator commands through the structure.

### 2.3.2. Design Issues and Metrics

This research investigates how the requirements of smart structures matches the capabilities of multicomputers. The way that the requirements and capabilities relate to smart structures and multicomputers individually was discussed in Section 2.1 and Section 2.2, respectively. This section briefly describes the issues that must be examined when considering the combination of smart structure and multicomputers.

2.3.2.1. Performance Requirements

The performance requirements of multicomputers for smart structure applications can be estimated by examining the processing requirements of each of the individual tasks. The first step is to determine how much processing is required for raw sensor measurements. Processing measurements are usually given in units of millions of instructions per second (MIPS) or millions of floating-point operations per second (MFLOPS). By multiplying the MIPS or MFLOPS required by a single sensor by the number of sensors in the structure, the total processing power for the initial level of processing can be determined.

The control or diagnosis algorithm determines how many operations are required to relate the sensor measurements, identify the current state of the structure, and generate the necessary output signals. The amount of additional processing needed to generate control signals depends on the number of actuators in the system and the amount of processing required by each actuator. Also, the amount of time that processors may be idle due to a lack of data should be considered. Specialized coprocessors such as vector or floating-point coprocessors may also be used, if needed. The decision of whether to use centralized or distributed processing can be made only after the processor power, the processor cost, the number of sensors and actuators in the system, and the candidate application algorithms have been examined.

Talat [2,3] claims that the location of the processors is determined by the computer architecture chosen, the topology of the network, and the ability to embed the processors. Graceful degradation of function is an important factor that is mentioned by Talat. For example, in military aircraft that are prone to in-flight damage, it is important for the health monitoring system to operate even after damage has been sustained. Damage that is sensed by the structure may cause some of the processors and communication links in a multicomputer to fail. Thus, the multicomputer should be able to reschedule processing and have alternate data paths to provide fault tolerance.

Several other issues affect communication performance in a smart structure. It is difficult to have an accurate global clock for large structures, so asynchronous communication must be used. Point-to-point communication links, shared communication busses, and different communication protocols may be used with possible advantages and disadvantages for each. Integrated smart structures may require that the control or diagnosis system share communication resources with other functions. Communication protocols can be modified to prioritize messages so that critical information is not delayed. The number and bandwidth of links must meet the traffic and performance requirement of the system.

#### 2.3.2.2. Design Constraints

An embedded computer system must exist harmoniously with its surroundings, so physical constraints must be considered in the design of a multicomputer for a smart structure. These constraints distinguish smart structures from other multicomputer applications.

The difficulties involved with embedding and connecting fibers in a composite are well known and were discussed in Section 2.1.2.1.2. These discussions, however, mainly concerned embedded fiber optic sensors. The same problems exist for fiber optic communication links in a multicomputer. These considerations add to the cost of the communication links.

The placement of processing nodes and communication links in the structure is also affected by physical concerns. The placement of sensors in the structure is determined by the strain fields in the structure that need to be monitored. Because it is desirable place the processors near the sensors, the sensor placement affects processor placement. In addition, the routes of the communication links in a smart structure are affected by the physical shape of the structure. Thus, the location of beams and panels that can carry the communication links must be considered when an interconnection network topology is chosen.

For typical multicomputers, the processing nodes are relatively close together and one power supply can be used for all nodes. In smart structures, power must be provided to potentially distant processors. Thus, a power distribution network will have to be developed and embedded into the structure to support the processing nodes. Such a power distribution scheme may also be necessary to support sensors and actuators.

Other, more subtle, physical constraints involve the dynamics of structures. The complexity of the control algorithms that are implemented by the multicomputer depends on the complexity of the model used to characterize physical interactions in the structure. Computing issues, such as the number of sensors and actuators that each processing node can control depend on the complexity of the control model. Thus, the partition sizes of parallel algorithms are dictated by physical constraints.

### 2.3.3. Hybrid Communication and Sensing

Hybrid sensing and communication has been proposed as a way to maximize the utilization of the embedded optical fibers in a system [104,105]. Simultaneous operation is possible since fiber optic communication systems tend to use only a small percentage of total bandwidth available on an optical fiber. The research discussed below summarizes hybrid sensing and communication schemes that may be feasible in smart structures.

The first research in the field of hybrid sensing and communication of which the author is aware was performed by Leung, *et al.* [106]. This research demonstrates the use of a multimode optical fiber for both communication and vibration sensing. The concept was later extended to single-mode fiber systems [107]. The experiments performed by Leung, *et al.* consist of sending a communication signal across a fiber optic link while a portion of the link is vibrated. For the multimode fiber experiments, the received optical signal is divided using a beam splitter and lenses, with one half of the signal sent to a communication receiver and the other half sent to a sensing receiver. The sensing receiver examines the speckle pattern generated by the output signal to determine the nature of the disturbance. For single-mode fiber, wavelength division multiplexing (WDM) is used to distinguish between sensing and communication signals. The communication signal is represented by a 1000 KHz sine wave, and results show that the signal-to-noise ratio (SNR) remains relatively constant as the frequency of the disturbance is changed. Both cases demonstrate that fiber optic links can be used as sensing devices without significantly degrading the quality of communication on the link.

Experiments by Fuhr, *et al.* [108] corroborate Leung's results. The hybrid system used in these experiments performs communication and vibration sensing on a multimode optical fiber attached to an aluminum beam. As in Leung's work [106], the speckle pattern of the output optical field is used to sense the vibration disturbances on the fiber from DC to 100 Hz. A 34 Mbps digital communication signal is used in these experiments and actual data, such as a digitized television picture, is transmitted over the link. Included with the digital communication system is an analysis of the bit error rate (BER) on the link. This analysis shows that the BER is not significantly affected by the vibration in the system as long as the transmitter power is set to an appropriate level. A drawback of this hybrid sensing and communication method is that sensor data processing requires too many calculations to be analyzed in real-time and thus would not be compatible with an application that requires a real-time control scheme.

Wiencko proposed concepts for hybrid sensing and communication networks [109,110]. He stresses the need in fiber sensor integration to understand the nature of the communication from sensors. Wiencko proposes modeling the action of the communication system as noise in the sensor system and the action of the sensor system as noise in the communication system. Modulation techniques such as time division multiplexing (TDM) and WDM are said to hold promise for hybrid systems. In addition to discussing techniques that could be used in hybrid systems, Wiencko discusses the relation of sensor multiplexing techniques to hybrid sensing and communication. Kersey [111] discussed theoretical mathematical models for performance analysis of a multiplexed sensor array. Wiencko extends the work of Kersey by proposing that a communication channel on an optical fiber can be treated like one of the sensing channels in the mathematical formulations.

### 2.3.4. Example Systems

From the discussion of requirements for smart structures and the capabilities of multicomputers, an experimental smart structure design can be envisioned. Both vibration suppression and health monitoring algorithms use distributed sensors. The physically parallel hierarchical algorithm shown in Figure 2.5 may be used. The initial level consists of processing nodes, placed near the sensors, that filter and analyze the sensor readings. This placement limits the length of the links that connect processors and sensors. After the sensor information is processed, it is passed to other nodes for higher level processing. Processing at this level monitors a region of the structure containing many sensors and characterizes the physical interactions that occur in the region.

Additional levels are added that interpret results from lower levels. Each higher level observes a larger portion of the structure until some single highest-level controller contains a complete description of the interactions that are occurring. The processing must be performed so that the amount of information that is passed to a higher level is less than the amount that was received by the lower level. In this way, the traffic can be limited as information is passed to higher levels.

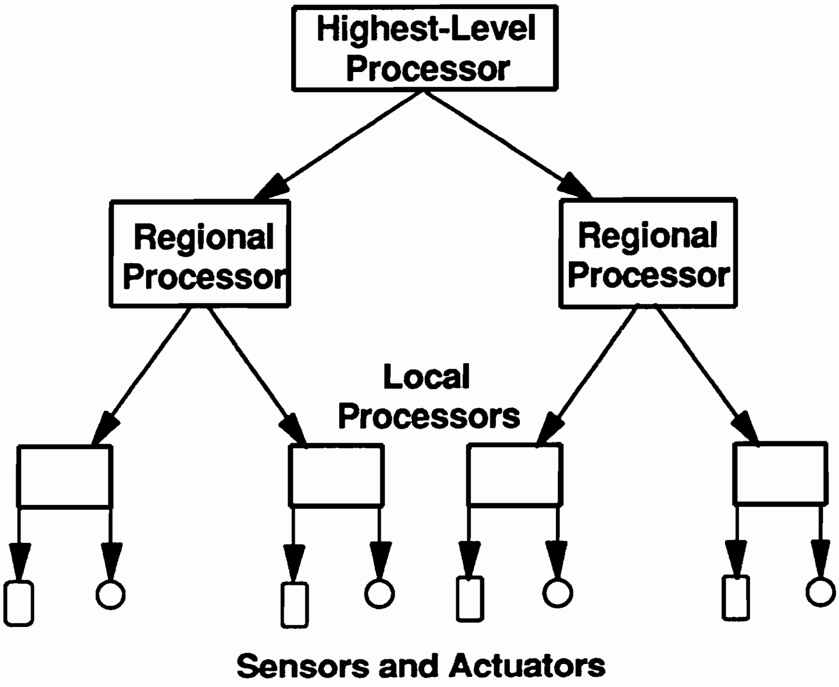


Figure 2.5. Physically parallel hierarchical network

At this point, the action of health monitoring and vibration suppression algorithms differ. In health monitoring applications, the highest level processor reports its observations concerning the system's overall action to an external source so that repairs and modifications can be made. For vibration suppression algorithms, the highest-level controller passes down information that can be acted on by the actuators. Just as information was passed up through hierarchy, information is passed down through the hierarchy to the actuators. At each level, a smaller picture of the controller's action can be gained until at the lowest level the processors issue control signals directly to the actuators.

As an alternative to this approach, a system could be created in which each node in the system performs functions from every level of the hierarchy. Such a system is shown in Figure 2.6. In this approach, every processor monitors and controls nearby sensors and actuators. It is responsible for processing the sensor measurements, sending control signals to the actuators, and informing surrounding nodes about its actions. Neighboring processors communicate on several different levels. At the lowest level, the processors share sensor data. At the next level, they share data concerning their view of the current system state. At the highest level, they share data concerning the control signals that are being sent out to nearby actuators. Thus, in this approach, each processor is assigned to a single region in the structure and is responsible for all of the actions that occur in that region.

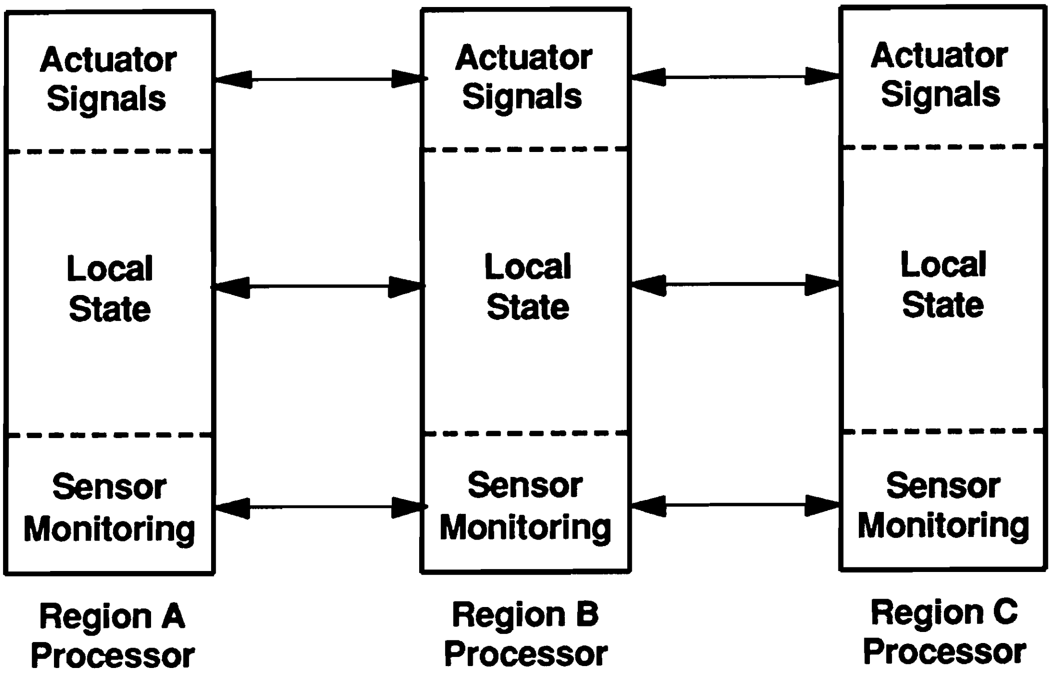


Figure 2.6. Algorithmically parallel hierarchical network

# Chapter 3. Objectives and Methodology

This chapter describes the objectives of this research and the methods that are used to achieve these objectives. The cost and performance metrics used to compare competing multicomputer implementations are discussed, and experiments that use these measures are described. Issues that are important to smart structure computer systems, but that lie outside of the scope of this research, are also presented.

## *3.1. Scope of this Research*

This research examines specific issues concerning multicomputers for smart structures. This section defines the issues that are examined. In addition, it discusses several parameters that are relevant to the study of multicomputers for smart structures, but fall outside of the scope of this work.

### 3.1.1. Goals

The overall objective of this research is to determine appropriate multicomputer architectures for smart structures. To achieve this objective, four subgoals are addressed. First, this research examines the suitability of multicomputers for distributed processing in smart structures. This goal is accomplished by integrating the findings of research in smart structures with research that examines multicomputers. This goal is also addressed by examining the operation of a three-processor testbed system and by performing simulation experiments using a damage detection, location, and estimation (DLE) algorithm for a ten-bay planar truss. The testbed system and the damage DLE algorithm simulations are described further in Sections 3.3.1 and 3.3.2, respectively.

The second goal of the research is to identify and define criteria that should be used to evaluate the effectiveness of multicomputer architectures for smart structures. This goal is satisfied by once again relating the findings of previous research in both smart structures and multicomputers. The cost metrics are the number of processing nodes and communication links in the network, the amount of embedded fiber required to implement the multicomputer in a structure, and the scalability of the architecture. The performance metrics include the time it takes to complete one iteration of a smart structure algorithm and the mean and standard deviation of message latency in the system. The cost and performance metrics are described more thoroughly in Sections 3.2.1 and 3.2.2, respectively.

The third goal of the research is to determine how algorithms executed in smart structures can be efficiently implemented on multicomputers. Prior work is used to determine the types of algorithms that will typically be executed in smart structures. Several algorithm characteristics are examined by the testbed and simulation experiments. The testbed system is used to study how the assignment of separate processes among the nodes affects the execution of the algorithm. Both the testbed and the simulation experiments demonstrate how the ratio of the communication time to the processing time can change the relative importance of the system parameters. The performance of a multicomputer algorithm is affected by background traffic in the network. The simulation experiments monitor the performance of the damage DLE algorithm as the background message generation rate is varied. By examining these factors, suitable characteristics for future algorithms are determined and methods for mapping smart structure algorithms onto parallel architectures are described.

The fourth goal of the research is to determine a set of characteristics that interconnection network topologies in smart structures should possess. A single topology cannot both maximize performance and minimize cost. This research examines several different topologies and determines how the inherent characteristics of each of the topologies affects performance. The simulation experiments monitor the behavior of three different interconnection network topologies, and the relative cost and performance of the topologies are compared. These comparisons identify the manner in which specific features of the topologies affect performance. Guidelines for choosing a topology for a specific application are also developed.

### 3.1.2. Limitations

Several important design issues fall outside the scope of this research. Fault tolerance is an important consideration in any large application. Multicomputers contain an inherent degree of fault tolerance because they contain multiple processors and, possibly, redundant communication paths between processors. Schemes to best exploit this fault tolerance are not considered in this research.

The electrical power needs of processors also present a problem for multicomputers in smart structures. Distributed power requirements also exist for the mechanical actuators in the structure, so the problem is not unique to processing components. Since the power consumption of multicomputer processing nodes will probably be smaller than the power required by the actuators, this problem should be addressed in conjunction with the actuation problem. Thus, power constraints are not considered in this research.

A final set of concerns that affects the performance of a multicomputer is the protocol and routing mechanism used for transmitting data on the communication links that connect the processing nodes. The first-come first-served message protocol and wormhole routing mechanism used in this research have been shown to be effective, but their use in this research is not meant to imply that they are the optimal choices for all multicomputers for smart structures.

## *3.2. Evaluation Criteria*

Before alternative multicomputer implementations can be compared, it is necessary to establish the criteria that form the basis of the comparison. This section describes cost metrics and performance measures that are used to determine the relative costs and benefits of competing multicomputer implementations.

### 3.2.1. Cost Metrics

This research evaluates the cost of a multicomputer implementation with respect to several different factors. Specifically, it examines the number of processors in the system, the number of communication links required by the interconnection network topology, and the amount of embedded fiber required to implement the system in a smart structure. In addition to evaluating these cost factors for a specific implementation, the scalability of these costs as the system size increases is also examined.

The first two cost metrics, the number of processors and the number of communication links, are commonly examined in multicomputer research [83]. The number of communication links in a topology gives a measure of the amount of support hardware such as line drivers and message buffers that are required [76]. The number of processing elements gives a direct indication of how much processing hardware is required. To obtain an exact cost estimate for the system, a detailed examination of factors such as the exact type and speed of the processors, the availability of coprocessor hardware, the data rate of the communication links, and the buffering hardware available for each link would be required. The simplified numerical measures examined in this research, however, provide an initial estimate of the complexity and cost of the hardware required to implement the system and are useful for the purpose of comparing alternative implementations.

The third cost metric examined in this research, the length of embedded fiber, is not typically considered in multicomputer evaluations. The communication links used in a smart structure multicomputer may need to be embedded in the composite material used to fabricate the structure. Embedded fiber in a composite material increases the complexity and cost of its manufacture [28,112]. An additional degree of complexity is created at the boundaries of elements in the structure where the communication links from one structural element are connected to links in a neighboring element. Thus, the cost of the communication links depends not only on the number of links, but also on the length of the links.

The physical size of smart structures and their processing systems vary from one application to the next. The ability to scale multicomputers to an appropriate size is desirable for smart structure applications. For this reason, this research examines how the cost metrics defined above change as the size of the structure for a specific application increases.

### 3.2.2. Performance Measures

Several multicomputer architectures are considered in this research. Each of the architectures is assumed to execute an iterative damage DLE algorithm that periodically examines the state of the structure. The performance of each multicomputer implementation is rated according to the time it takes to complete a single iteration of the algorithm and the message delay incurred in the system.

The time required to complete a single iteration of the algorithm, also referred to as the cycle time, indicates how often the algorithm can be executed, and thus defines the maximum rate at which information about the state of the structure can be updated. The cycle time is affected by the operation of the processing system, the operation of the communication network, and the interaction of the two. By determining the cycle time for the experimental architectures examined in this study, and by examining changes in the cycle time as system parameters are varied, it is possible to better understand how the characteristics of the architectures affect system operation.

An emphasis of this research is to study which communication topologies are best suited for smart structures. Thus, in addition to examining the overall system operation, the operation of the communication network is examined in detail. In particular, statistics are collected for the mean and standard deviation of message delay in the system. These statistics are categorized by splitting up the statistics for broadcast and non-broadcast transmissions and by separating the delay statistics for messages that are generated by the algorithm from the statistics for background messages in the network. The real-time nature of smart structure applications requires that the communication between the nodes be executed predictably and with low latency. Thus, information concerning the mean and variance of message delay provide insight into the effectiveness of the architecture for real-time applications.

### *3.3. Experiments*

The cost metrics and performance measures described in the preceding section are used to compare several architectures. Initial experiments were performed on a small testbed network. Simulation models of multicomputer systems were then developed to examine the performance of a damage DLE algorithm for a large truss structure. The testbed and simulation experiments are briefly outlined below.

### 3.3.1. Testbed Network

The first experiment examined in this research is performed on is a three-processor testbed network that periodically updates a shape estimate of a triangular beam structure subjected to external forces. The shape estimation procedure is divided into several processing tasks. Three different organizations of the processing tasks are considered. All three organizations perform essentially the same function, but they differ in the manner in which they assign the various processing tasks to the processing nodes. The operation of the three processors is monitored, and the performance of each of the schemes is determined.

All of the organizations use the same hardware, so cost comparisons for these implementations are not an issue. Also, the communication times in the network are much smaller than the processing times required by the algorithms, so comparisons of the message passing delay are also not considered. Instead, the testbed is used to examine the cycle time of different algorithms that execute on the same architecture. The testbed algorithms are executed both with and without math coprocessors in the processing nodes so that the effects of the ratio of processing time to communication time on the cycle time of each of the organizations can be monitored. The testbed provides insight into the problem of mapping an algorithm onto a specific architecture and illustrates the effects that data dependencies can have on system performance. The testbed also serves as a physical example of a smart structure implementation and provides timing data that is used in the formulation of simulation models of larger architectures. A complete description of the testbed network and the results that were obtained is presented in Chapter 4.

### 3.3.2. Damage DLE Algorithm Simulation

A damage detection, location, and estimation algorithm is presented as a typical algorithm executed on a smart structure multicomputer. The operation of the algorithm is simulated for an application involving a ten-bay truss. The effects of modifications to the network are examined by changing parameters in the simulation model. Cost metrics and performance measures are evaluated for each set of simulation parameters, and the relevance of the results to the application of multicomputers for smart structures are discussed.

Three parameters are examined in detail for the simulations. The first parameter is the topology of the interconnection network. Three different topologies are simulated. The characteristics of each of the topologies are discussed and the manner in which these characteristics affect the cost metrics and performance measures of the simulation are observed.

Another simulation parameter is the rate at which background message traffic is generated in the network. The background messages compete with algorithm-generated messages for communication resources and thus affect the ability of the network to execute the damage DLE algorithm. The effect of the background message rate is measured by examining how the performance measures defined in Section 3.1.2 change as the background message generation rate varies.

The final parameter examined in the simulations is the ratio of communication time to processing time. In one case, the processing time is set to be equal to the communication time. In the other cases, the processing time is set to be either much larger or much smaller than the communication time. In this way, the system operation is examined for the case where the processing system is the bottleneck and for the case where the communication system is the bottleneck. The performance of the system is determined for each of the ratios of communication time to processing time and reasons for performance differences are discussed.

A thorough description of the damage DLE algorithm, as well as a discussion of the truss structure and the interconnection network topologies under consideration, are contained in Chapter 5. Chapter 6 describes the VHDL model that is used to simulate the experimental architectures and discusses the results of the simulation experiments.

# Chapter 4. The Testbed System

This chapter describes an experimental distributed processing application that determines the physical state of a small triangular structure. The chapter begins with a brief description of the physical parameters of the structure and the fiber optic sensors and communication links that were used. The emphasis of the chapter, however, is on different algorithms used to calculate the state of the structure. The discussion of each algorithm examines the following factors: the manner in which the computational workload is partitioned among the three processing elements, the communication that is required between the processing elements, and the relative timing of processes executed by separate processing nodes. In addition to describing the accuracy of the system in determining the physical state of the structure, the chapter also discusses the total processing time and relative performance of the algorithms.

## *4.1. The Experimental Setup*

This section describes individual functions of the experimental system and their implementation. The description is split into two areas. First, the physical layout of the structure and the mathematical calculations used to relate the strain measurements obtained by the fiber optic sensors to the state of the structure are described. Next, a hybrid communication/sensing network that measures strain in the structural elements, shares data between processing nodes, and calculates the overall state of the structure is presented.

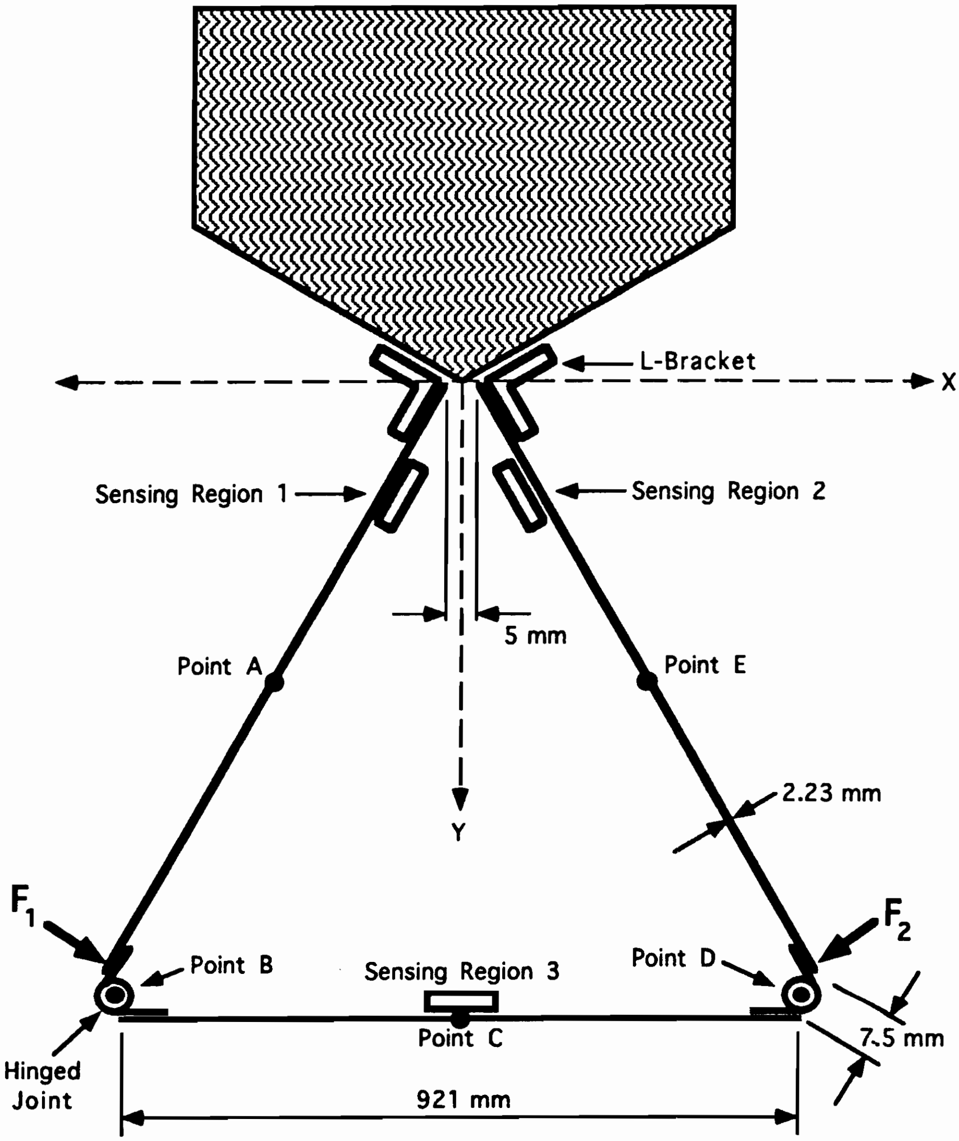
### 4.1.1. Physical and Mathematical Description of the Structure

A diagram of the top view of the test structure used for this research is shown Figure 4.1. Beams 1 and 2 are clamped to a stable base using two L-brackets. The two ends of beam 3 are joined to the unclamped ends of beams 1 and 2 using a hinged joint as shown in the figure.

Although the beams are clamped to a stable base, they are flexible in the X-Y plane. Thus, forces applied to the unclamped ends of beams 1 and 2 cause the structure to deform. The extent of the deformation depends on the magnitude and direction of the applied forces. The intent of the experiment is to sense and monitor the deformation in the structure for forces, labeled $F_1$ and $F_2$ in the figure, applied perpendicularly to the unclamped ends of the side beams.

The state of the structure is predicted by sensing the strain in the structure at several locations. A sensing region is defined for each of the beams that monitors the deformation of the beams in response to the previously defined forces. The maximum strain of a clamped-free beam subjected to a single point force occurs at the clamped end of the beam. Thus, the sensing region for each of the two side beams is placed near the clamped end of the beam. The ends of beam 3 are attached to the side beams with a hinged joint that enables the ends of beam 3 to rotate and counteract strain created by the indirect forces received from the side beams. Thus, the maximum strain for beam 3 is located at the center of the beam, and the third sensing region is placed at this location.

The calculations required to predict the location of the beams are performed after the structural model is decomposed into individual parts. As shown in Figure 4.2, beam 1 is



Beam Length: 921 mm
Beam Height: 31.5 mm
Beam Width: 2.23 mm

Figure 4.1. The test structure

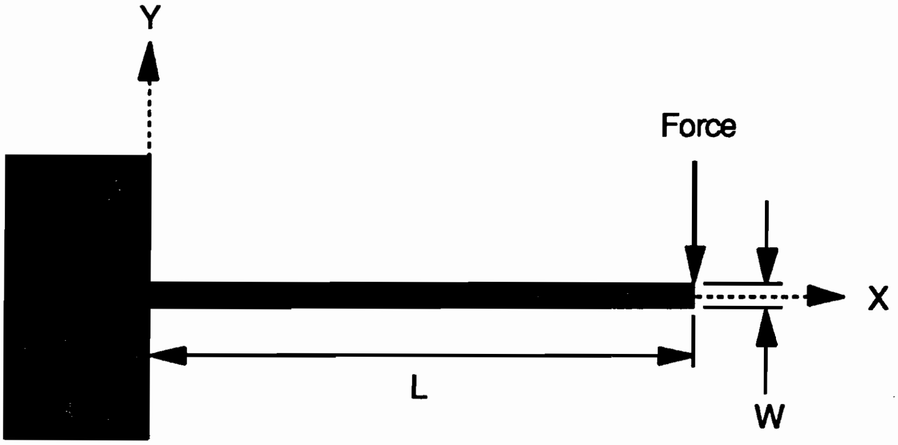


Figure 4.2. Cantilevered beam with an applied external force

modeled as a cantilevered beam with a force applied to the end. The sensor is assumed to measure the strain at the base of the beam. Also, the deflection of the beams is assumed to be small. With these conditions, the deflection of the beam in the $y$ direction for a point on the beam $x$' meters from the base is given by

$$y(x') = \frac{2\epsilon(0)}{\mathrm{WL}} \, [\, - (\mathrm{L}-x')^3 + 3\mathrm{L}^2(\mathrm{L}-x') - 2\mathrm{L}^3 \,] \tag{1}$$

where L is the length of the beam in meters, W is the width of the beam in meters, and $\epsilon(0)$ is the strain at the base of the beam [113]. To more accurately account for the shortening of the x direction that is caused by deflections in the $y$ direction, the following formula is used to calculate the $x$ coordinate once the $y$ coordinate has been calculated.

$$x = (\, x'^2 - y^2 \,)^{1/2} \tag{2}$$

The final coordinates for points along beam 1 are calculated by rotating the results achieved in the single cantilevered beam model by 120° to account for the position of beam 1 with respect to the origin of the structure. Beam 2 is modeled using equations similar to those used for beam 1 except that the results are rotated by 60° instead of 120°.

The end points that are calculated for beams 1 and 2 are used to begin the calculations for the position of beam 3. The distance between points 1 and 2 is used as the length of the beam. As shown in Figure 4.3, beam 3 is modeled as a pinned-sliding pinned beam. Once again, a small deflection model is used and the deflection in the $y$ direction for a point which is a distance $x$' from the pinned end, where $x$' is between 0 and $\frac{\mathrm{L}}{2}$ is given by [113]

$$y(x') = \frac{2\epsilon(\frac{\mathrm{L}}{2})}{\mathrm{WL}} \, [\frac{x'^3}{3} - \frac{\mathrm{L}^2}{4} x'] \,. \tag{3}$$

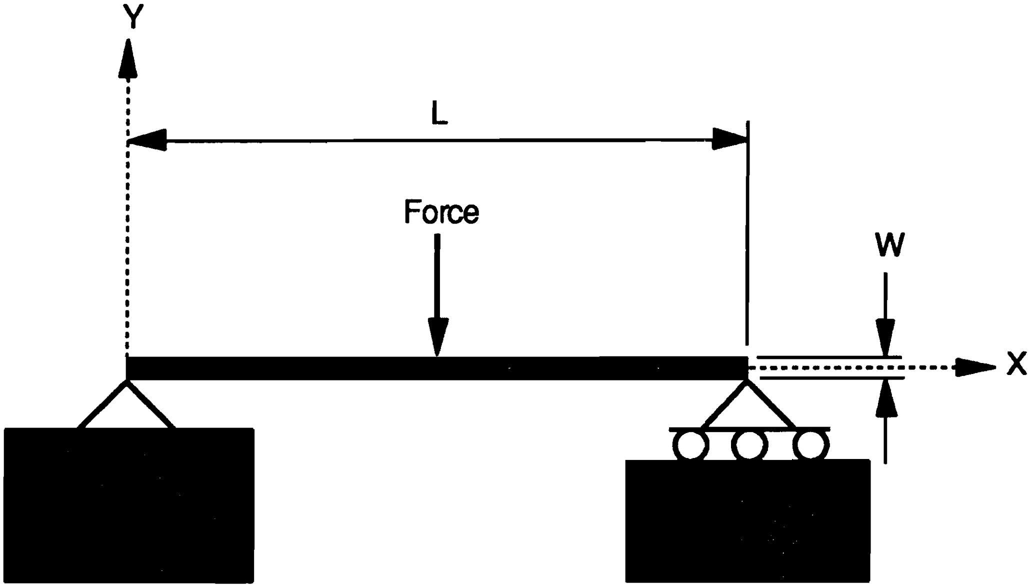


Figure 4.3. Pinned - sliding pinned beam with an applied external force

The deflection of the beam is symmetric about $\frac{L}{2}$, so the deflection for a value of $x$' between $\frac{L}{2}$ and L is calculated by the equation for $y(L\text{-}x')$.

The results for the models of beams 1 and 2 are simply rotated around the origin to find the final position of the beams in the structure, but the results for the model of beam 3 must be superimposed on the endpoints of beams 1 and 2. This is done by allowing the endpoint of beam 1 to serve as the origin, calculating the angle between the x axis and the line connecting the endpoints of beams 1 and 2, and then rotating the results achieved in the Equation 3 by this angle.

Although identifying the end points of the beams is significant, a better picture of the structure is given by calculating the position of several points along the beam. Thus, the position of points along the beams are calculated for the midpoints of the beams in addition to identifying the location of the end points of the beams.

### 4.1.2. Hybrid Sensing and Communication Network

A hybrid sensing and communication network, shown in Figure 4.4, provides the communication, sensing, and computation capabilities required for monitoring a small structure [105]. The testbed network consists of three personal computers that serve as processing nodes. Nodes 1 and 2 are IBM PS/2 Model 30s with 8 MHz 8086 processors. Node 3 is an IBM PS/2 Model 30/286 with a 10 MHz 80286 processor. Each node contains a communication transmitter, a communication receiver, and a sensing receiver.

Although each of the transmitters and receivers is connected to a separate fiber optic link, the 4-by-4 transmissive star coupler makes the network operate like a bus. The signal transmitted by each of the transmitters is received by all of the receivers. Wavelength division multiplexing (WDM) is used to isolate the effects of the communication and sensing signals. This allows the communication and sensing subsystems to operate independently.

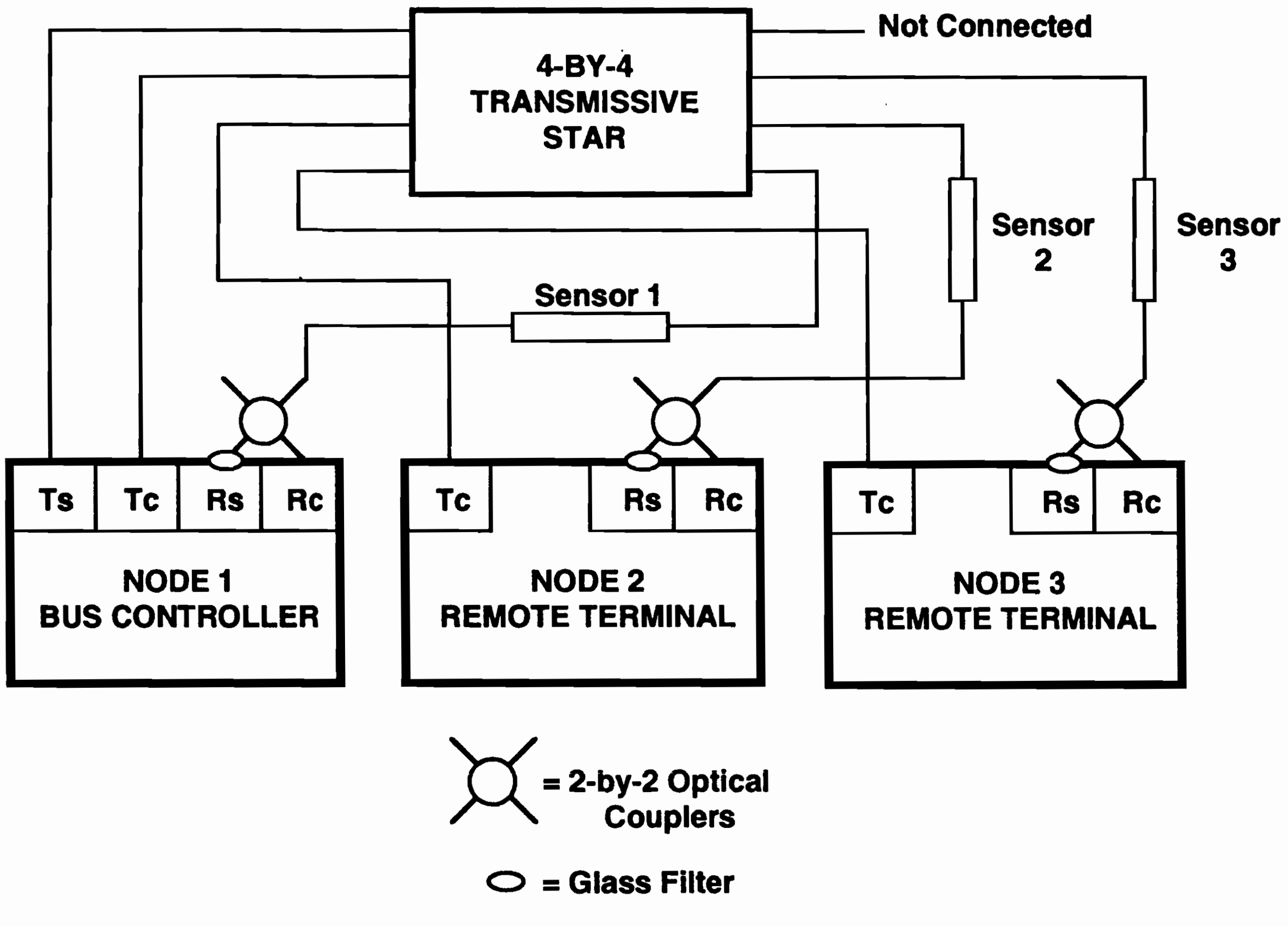


Figure 4.4. Hybrid sensing and communication network

| | |
|---|---|
| Ts = sensing transmitter | Tc = communication transmitter |
| Rs = sensing receiver | Rc = communication receiver |

#### 4.1.2.1. MIL-STD-1773 Communication

A control module is used in each of the processing nodes. Part of this module acts as a communication controller. It serves as a buffer between the central processor and the communication network. Data transmitted by the node are passed to the communication controller. The communication controller formats the data in accordance with the MIL-STD-1773 protocol and transmits data words on the link. The controller sends an interrupt signal to the central processor when transmission is complete. Similarly, when the node receives data over the communication link, the central processor is interrupted and the data are transferred from the buffer to the central processor. In this way, the central processor only deals with the communication system when it sends a message to the control module or when it is interrupted by the communication controller when a message transfer is completed. During the time when the message is being transmitted on the link, the central processor is available to perform other processing tasks.

MIL-STD-1773 is a standardized fiber optic communication protocol that is derived from the MIL-STD-1553 electrical bus protocol [114, 115]. It operates as a 1 Mbps digital communication link in which a Bus Controller (BC) shares data with up to 31 Remote Terminal Units (RTUs). Because the fiber optic links act like a single bus, only one node may transmit on the link at a time. Link arbitration is performed by the BC. When a data transfer is required between two RTUs, the BC sends control commands that set up the transmission and gives one of the RTUs permission to transmit on the link. In addition, the BC itself may share messages with the RTUs. Each message may contain up to 32 words of data.

#### 4.1.2.2. Strain Sensing

While part of the control module serves as a communication controller, the remainder of the control module contains the sensing subsystem. The sensing subsystem uses intensity-based sensing methods [105]. A constant intensity light is coupled into the network by the sensing transmitter. This optical energy is divided by the 4-by-4 transmissive star coupler and distributed equally among the three output fibers that lead to the sensing receivers. As the light travels to the sensing receivers, it goes through a sensing region in which the fiber is cut and placed into a hollow tube was shown in Figure 2.1. The hollow tube is then attached to the sensing region of a beam with a gap separating the two ends of the fiber in the tube. As the beam bends, the length of the gap changes. This change in gap length, in turn, changes the intensity of the light at the output of the fiber. A detector is used at the output of the fiber that converts the light intensity to an analog electrical signal.

After the electrical signal is passed through a series of amplifiers and filters, an analog-to-digital converter converts it to a digital value. This digital value, which can be read by the central processor through the control module, is directly related to the length of the gap separating the ends of the fibers in the hollow tube which, in turn, is related to the strain of the beam. The exact relationship between strain and the digital value produced by the sensor depends on many factors such as the gauge length of the sensor, the location of the sensor, the optical attenuation that occurs in the fiber optic links, and the amplification of the electrical signal.

Instead of calculating the effects of each of these factors, the sensor is calibrated by recording the physically measured strain and the digital value produced by the sensor for different beam positions. Curve fitting is then applied to this data and a third order polynomial function that relates strain in the beam to the digitized sensor value is produced and coded in the program at each node.

#### 4.1.2.3. Process Execution Monitoring

For experimental purposed, the processors are monitored as they perform calculations. The operation of the distributed network is decomposed into independent subprocesses, and a four-bit identification value is assigned to each subprocess. The first statement executed by each subprocess outputs the four-bit identification value of the process through the parallel port of the computer. A logic analyzer monitors the identification value output by each processor and produces traces that indicate the relative timing of process execution on the three processing nodes.

## *4.2. Application Parameters*

The application discussed in this chapter uses the hybrid sensing and communication network to repeatedly determine the shape of the beams as various external forces are applied to the structure. This is done by examining the sensor measurements and performing the calculations described in Section 4.1.1. The results of the calculations are reported by one of the processing nodes. This node draws the current shape of the structure on a video monitor. In an alternate configuration, the node also writes the coordinates of the structure to a data file.

A key component of this application is the ability of the system to update the shape coordinates at consistent time intervals. An update cycle is created by defining a time-ordered list of processes required for each update. The start of each update cycle is signaled by a timer. During each update cycle, the sensor values are sampled, data is transmitted between the processing nodes, the coordinates of points on the structure are calculated, and the shape of the structure is drawn on the screen. The time it takes to complete one update cycle must be smaller than the interval between update cycles. Thus, the state of the structure can be updated more frequently as the update cycle processing time decreases.

This section describes several different methods for distributing the processes among the processing elements. The different processes that are executed are defined. Also, this section discusses how the study examines the effects of decreasing execution time, by using math coprocessors, on system performance.

### 4.2.1. Process Descriptions

Table 4.1 lists the processes that are executed and their corresponding identifier. The number of processes and the order in which they are executed depends on the algorithm used for state computation. Not all of the processes are necessarily executed by a given algorithm. Some of the algorithms require only a subset of these processes to calculate the state of the structure.

### 4.2.2. Processing Methods

The performance of the any processing system depends on the algorithm it is executing. Three different algorithms were examined for the test structure. The distinctions between these processing methods are largely dependent on three factors: the assignment of the communication control process, the method of computing the data points, and the assignment of the display process.

The bus controller in the MIL-STD-1773 protocol controls data sharing between the processors. Thus, processing time is required from the node that acts as the BC each time any two processors want to communicate. The BC is also responsible for sending the synchronization pulse and for scheduling data flow between processes. This means that the BC node must initialize and monitor a timer so that the sampling periods commence at consistent intervals. These control functions add an additional processing burden to the node that acts as the BC.

Table 4.1. Process definitions

| Identification Code | Process Description |
|---|---|
| 0 | Execution terminated |
| 1 | System initialization |
| 2 | Synchronization |
| 3 | Waiting for timer interrupt |
| 4 | Acquiring data for beam 3 |
| 5 | Acquiring data for beam 1 |
| 6 | Acquiring data for beam 2 |
| 7 | Transmitting/Receiving all coordinates |
| 8 | Calculating points for beam 1 |
| 9 | Calculating points for beam 2 |
| 10 | Calculating points for beam 3 |
| 11 | Drawing the screen |
| 12 | RTU idling |
| 13 | RTU saving next sample |
| 14 | Communication interrupt service |
| 15 | Timer interrupt service |

Using centralized computation, data from the three sensors are transmitted to a single processor. This processor then performs all of the calculations discussed in Section 4.1.1. In the parallel computation method, the processors that monitor the side beams receive their local sensor reading and compute the coordinates of the corresponding beam. They then transmit these coordinates to the processor that monitors beam 3. This processor uses the coordinates from the other processors and the sensor measurement from beam 3 to determine the coordinates for beam 3.

The final major process is reporting the results of the computations. In some applications, these results might be stored so that the state of the structure can be analyzed at a later date. Other systems may examine the data in real-time and signal an operator when problems are identified. Also, the current state of the structure could be output on a display terminal and updated after each sample period. In the experiment discussed in this chapter, the calculated structural configuration is drawn on the display of one of the processing nodes. The drawing routines invoke graphics commands that require a significant amount of time to execute. Thus, the processing node that is assigned the task of drawing the current state of the structure must devote a significant portion of time to reporting the calculated results. A flowchart for the processing tasks executed in these experiments is shown in Figure 4.5. The different schemes for organizing the processing tasks are described below.

#### 4.2.2.1. Centralized Computation with Combined Control and Display

The first algorithm attempts to use a single processor for all tasks. Because each of the sensor measurements is monitored on a separate processor, it is not possible to completely eliminate the other processors. Wherever possible, however, processes are assigned to the 80286 processing node.

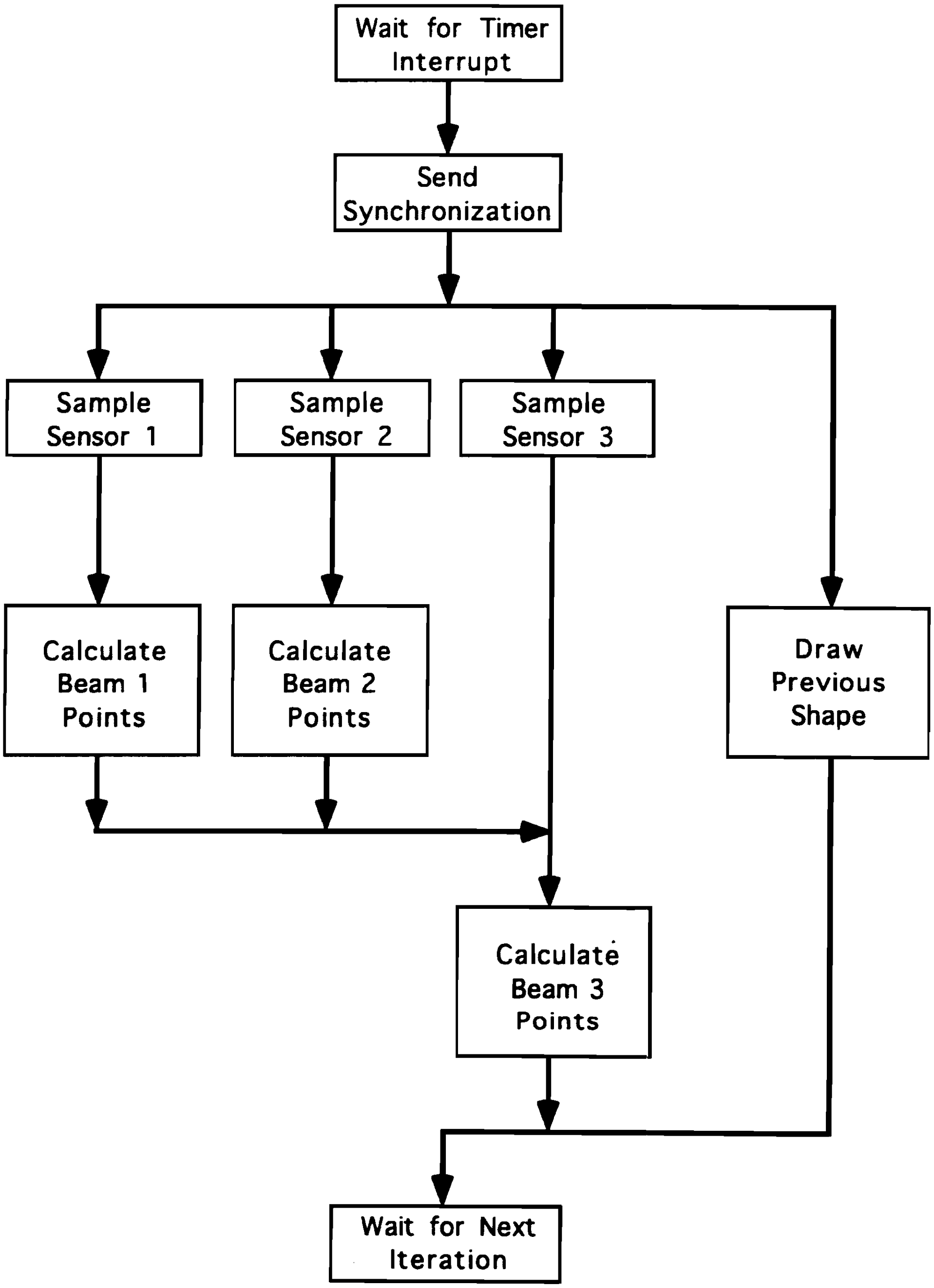


Figure 4.5. Flowchart of the shape monitoring algorithm

The 80286 acts as the BC and begins the processing cycle by issuing a synchronization command that causes the other processors to sample their sensors. The 80286 then reads its own sensor measurement and receives a sensor measurement from each of the two RTUs. All of the beam coordinates are calculated from these sensor measurements. After calculating these coordinates, the state of the structure is drawn on the screen. When this cycle is completed, the 80286 waits for a timer interrupt that signals that a new processing cycle should begin.

#### 4.2.2.2. Parallel Computation with Combined Control and Display

Instead of having a single processor perform all of the calculations, the calculations of the beam coordinates can be distributed among the three processors. Because the 80286 processor is faster than the other two machines, it is initialized as the BC and is also assigned the task of drawing the state of the system on the screen. The cycle begins with the BC issuing a synchronization command. After synchronization, each processor samples its local sensor. The 80286 processor then begins to draw the previous system state on the screen, while the other two processors calculate the coordinates of their corresponding beam and send their results to the communication controller. When the BC is finished drawing the previous state of the structure, it commands both RTUs to transmit the beam coordinates that they have calculated. The BC then calculates the coordinates for beam 3 based on the its local sensor measurements and the coordinates received from the RTUs. The BC then waits for the timer interrupt that signals it to perform operations for the next sample period.

#### 4.2.2.3. Parallel Computation with Split Control and Display

The final algorithm requires the most interprocessor communication, but it more evenly distributes the tasks among the processing nodes. The most time consuming process is the task of drawing the system state on the screen. Thus, this task is assigned to the 80286 processor.

Because the 80286 is busy with the task of drawing the screen, the 8086 machine that previously served as RTU 2 is initialized as the BC. The other 8086 machine serves as RTU 1 while the 80286 serves as RTU 3.

The cycle begins with the BC issuing the synchronization command. Each of the three processors sample their local sensor. The 80286 machine sends its sensor value to its communication controller so that it can be accessed by the BC. Immediately after saving its sensor value, the 80286 begins to draw the state of the structure calculated during the previous sample period.

While the 80286 is drawing the screen, the two 8086 machines use their local sensor values to compute the coordinates of their corresponding beam. The BC then commands RTU 3 to send its sensor value and commands RTU 1 to send the coordinates it calculated. After sending the data coordinates for beam 1, RTU 1 waits for the next sample period to commence. The BC and RTU 3 still have tasks to perform. After receiving data from the two RTUs, the BC computes the coordinates of beam 3. At this point, the BC has the coordinates of all three beams. The BC concludes the cycle by sending these coordinates to RTU 3. It then waits for the timer interrupt that signals it to begin the next cycle period.

When RTU 3 completes the screen drawing process, it waits to receive the next set of state coordinates from the BC. After receiving the coordinates, it stores them so that they can be used to draw the screen in the next sample period and then waits for the synchronization command that signals the beginning of the next sample period.

### 4.2.3. Math Coprocessor Option

The calculations performed in this application consist of many mathematical functions whose computation times can be greatly reduced by using a math coprocessor in addition to the main processor. The 8086 machines use a 8087 math coprocessors, and the 80286 machine uses an 80287 math coprocessor. The timing of each of the three processing methods described above is examined for a system using and for a system not using math coprocessors. From this examination, it is possible to determine how changes in the duration of subprocesses affect the performance of each of the three algorithms.

## *4.3. Results*

The results are discussed in two sections. The first set of results describes how accurately the model was able to predict the physical state of the structure. The second set examines how the different processing algorithms and computation methods affect the system timing.

### 4.3.1. Model Accuracy

As shown in Figure 4.1, five points, labeled A through E, are marked on the structure. For each configuration, the coordinates of these points were physically measured while the test system also calculated the position of these points based on the sensor measurements. The coordinates of the measured data points are shown in Table 4.2, and the coordinates of the calculated data points are shown in Table 4.3. Since five data points were measured for each of the fifteen configurations, a total of seventy-five data points were examined. The measured values were compared to the calculated values for each of these seventy-five samples.

#### 4.3.1.1. Comparison of Observed Data Points and Measured Points

The first statistics calculated from this data, as shown in Table 4.4, are the distances between the measured and calculated coordinates. Averaged over all of the data points, the mean distance is 3.3 cm. A better understanding of the error can be gained by examining the mean error for each of the five data points shown in Table 4.5. For points A and E, which typically are about 46 cm from the origin, the average errors are 1.9 cm and 1.0 cm,

Table 4.2. Measured coordinates (in centimeters)

| $X_A$ | $Y_A$ | $X_B$ | $Y_B$ | $X_C$ | $Y_C$ | $X_D$ | $Y_D$ | $X_E$ | $Y_E$ |
|---|---|---|---|---|---|---|---|---|---|
| -24.1 | 39.3 | -47.5 | 79.1 | -0.6 | 79.4 | 45.6 | 79.8 | 22.7 | 39.7 |
| -26.5 | 37.8 | -55.2 | 73.9 | -9.0 | 78.7 | 37.3 | 83.8 | 20.2 | 41.0 |
| -29.1 | 35.7 | -63.9 | 66.1 | -18.3 | 76.4 | 26.3 | 87.4 | 17.1 | 42.2 |
| -22.0 | 40.5 | -40.0 | 82.9 | 6.6 | 79.4 | 52.2 | 75.3 | 24.9 | 38.3 |
| -19.8 | 41.6 | -32.9 | 85.7 | 13.1 | 78.4 | 58.2 | 70.1 | 26.9 | 37.0 |
| -18.2 | 41.9 | -28.3 | 87.1 | 17.6 | 77.6 | 62.2 | 66.8 | 28.1 | 35.9 |
| -17.9 | 42.1 | -30.1 | 86.3 | 16.1 | 82.7 | 60.0 | 69.0 | 27.1 | 36.6 |
| -19.3 | 41.4 | -35.8 | 84.3 | 10.0 | 88.0 | 53.7 | 74.0 | 24.9 | 38.3 |
| -19.8 | 41.4 | -35.9 | 84.5 | 9.0 | 73.6 | 55.1 | 73.1 | 25.6 | 37.9 |
| -31.3 | 33.6 | -69.3 | 59.2 | -24.7 | 72.8 | 18.9 | 89.0 | 14.0 | 43.0 |
| -26.9 | 37.4 | -57.1 | 71.8 | -11.0 | 78.1 | 34.7 | 84.8 | 19.7 | 41.3 |
| -19.4 | 41.6 | -30.5 | 86.2 | 15.1 | 77.9 | 60.8 | 68.7 | 27.4 | 36.4 |
| -26.5 | 37.8 | -55.2 | 73.9 | -9.0 | 78.7 | 37.3 | 83.8 | 20.2 | 41.0 |
| -22.0 | 40.5 | -40.0 | 82.9 | 6.6 | 79.4 | 52.2 | 75.3 | 24.9 | 38.3 |
| -26.9 | 37.4 | -57.1 | 71.8 | -11.0 | 78.1 | 34.7 | 84.9 | 19.7 | 41.3 |

Table 4.3. Calculated coordinates (in centimeters)

| $X_A$ | $Y_A$ | $X_B$ | $Y_B$ | $X_C$ | $Y_C$ | $X_D$ | $Y_D$ | $X_E$ | $Y_E$ |
|---|---|---|---|---|---|---|---|---|---|
| -25.7 | 38.2 | -54.1 | 74.5 | -3.5 | 76.5 | 47.0 | 79.2 | 23.4 | 39.7 |
| -28.1 | 36.5 | -60.1 | 69.1 | -11.0 | 75.7 | 38.8 | 83.6 | 20.6 | 41.2 |
| -30.6 | 34.3 | -68.2 | 61.9 | -20.3 | 74.2 | 27.2 | 88.0 | 16.9 | 42.8 |
| -23.6 | 39.6 | -47.7 | 78.8 | 3.6 | 76.3 | 54.8 | 74.0 | 26.0 | 38.0 |
| -21.4 | 40.8 | -41.1 | 82.4 | 9.4 | 76.4 | 59.9 | 70.0 | 27.7 | 36.8 |
| -19.8 | 41.6 | -36.3 | 84.7 | 13.6 | 76.1 | 63.3 | 66.9 | 28.9 | 35.8 |
| -19.0 | 42.0 | -33.7 | 85.7 | 15.2 | 81.0 | 62.4 | 67.7 | 28.6 | 36.1 |
| -20.9 | 41.0 | -39.6 | 83.2 | 9.4 | 89.8 | 56.0 | 73.2 | 26.4 | 37.8 |
| -20.8 | 41.1 | -39.4 | 83.3 | 8.4 | 70.0 | 58.1 | 71.5 | 27.1 | 37.2 |
| -33.2 | 31.7 | -75.3 | 53.0 | -26.0 | 68.4 | 21.0 | 89.7 | 14.9 | 43.5 |
| -28.8 | 35.9 | -63.0 | 67.2 | -12.5 | 74.8 | 37.7 | 84.0 | 20.3 | 41.3 |
| -21.1 | 40.9 | -40.3 | 82.8 | 11.2 | 75.0 | 62.8 | 67.4 | 28.7 | 36.0 |
| -28.2 | 36.4 | -61.3 | 68.8 | -10.6 | 75.2 | 39.8 | 83.1 | 20.9 | 41.0 |
| -23.7 | 39.5 | -48.0 | 78.6 | 3.9 | 75.6 | 55.9 | 73.2 | 26.3 | 37.8 |
| -28.8 | 35.9 | -62.9 | 67.3 | -12.3 | 74.5 | 38.0 | 83.9 | 20.4 | 41.3 |

**Table 4.4.** Distance between measured and calculated coordinates (in centimeters)

| $X_A$ | $Y_A$ | $X_B$ | $Y_B$ | $X_C$ | $Y_C$ | $X_D$ | $Y_D$ | $X_E$ | $Y_E$ |
|---|---|---|---|---|---|---|---|---|---|
| -1.7 | -1.1 | -6.6 | -4.5 | -2.8 | -2.8 | 1.4 | -0.6 | 0.6 | 0.0 |
| -1.6 | -1.4 | -4.9 | -4.8 | -2.0 | -3.0 | 1.4 | -0.2 | 0.3 | 0.2 |
| -1.5 | -1.3 | -4.4 | -4.2 | -1.9 | -2.2 | 0.9 | 0.6 | -0.2 | 0.5 |
| -1.6 | -0.9 | -7.6 | 4.1 | -3.0 | -3.2 | 2.6 | -1.3 | 1.1 | -0.3 |
| -1.6 | -0.8 | -8.2 | -3.3 | -3.7 | -2.1 | 1.7 | -0.5 | 0.8 | -0.2 |
| -1.6 | -0.4 | -8.0 | -2.5 | -4.0 | -1.5 | 1.1 | 0.1 | 0.9 | -0.1 |
| -1.1 | -0.1 | -3.1 | -0.6 | -0.9 | -1.7 | 2.4 | -1.3 | 1.5 | -0.6 |
| -1.6 | -0.4 | -3.8 | -1.1 | -0.6 | 1.8 | 2.2 | -0.8 | 1.5 | -0.6 |
| -1.1 | -0.3 | -3.5 | -1.2 | -0.6 | -3.6 | 2.9 | -1.7 | 1.5 | -0.7 |
| -1.9 | -1.8 | -6.0 | -6.1 | -1.3 | -4.4 | 2.1 | 0.6 | 0.9 | 0.5 |
| -1.9 | -1.5 | -0.6 | -4.7 | -1.5 | -3.4 | 3.0 | -0.7 | 0.6 | 0.1 |
| -1.7 | -0.7 | -9.8 | -3.4 | -3.9 | -3.0 | 2.0 | -1.4 | 1.3 | -0.4 |
| -1.7 | -1.5 | -6.1 | -5.1 | -1.7 | -3.5 | 2.4 | -0.7 | 0.7 | 0.0 |
| -1.7 | -1.0 | -8.0 | -4.4 | -2.7 | -3.8 | 3.7 | -2.1 | 1.4 | -0.5 |
| -1.9 | -1.4 | -5.7 | -4.5 | -1.3 | -3.6 | 3.2 | -1.0 | 0.7 | 0.0 |

Table 4.5. Error statistics (distances in centimeters)

| Point | Mean Error | Std. Dev. of Error | X Correlation | Y Correlation |
|---|---|---|---|---|
| A | 1.9 | 0.44 | 0.97 | 0.96 |
| B | 7.2 | 2.13 | 0.96 | 0.96 |
| C | 3.8 | 0.90 | 0.96 | 0.95 |
| D | 2.4 | 0.91 | 0.96 | 0.96 |
| E | 1.0 | 0.43 | 0.96 | 0.96 |

respectively. For points B and D, which are typically about 92 cm from the origin, the average errors are 7.2 cm and 2.4 cm, respectively. For point C, which is typically 80 cm from the origin, the mean error is 3.8 cm.

In addition to presenting the mean error for each of the five data points, Table 4.5 also lists the standard deviation of the error. For points A and E, the standard deviations of the error are 0.44 cm and 0.43 cm, respectively. For points B and D, they are 2.13 cm and 0.91 cm, respectively, and for point C it is 0.90 cm. Thus, the standard deviation of error is less than 1 cm for every point except point B. This shows that although the average error is as high as 7.2 cm, the magnitude of the error is somewhat consistent.

Table 4.5 also lists the correlation between the measured coordinates and the calculated coordinates. This measure describes the consistency with which the calculated coordinates change in reaction to a change in the measured coordinates. A value of 0 means that the two sets of data are completely uncorrelated, while a value of 1.0 means that their is a perfect correlation between the calculated and measured data. A correlation of 0.95 or greater is achieved for both the x-axis and y-axis coordinates of each of the five data points. This shows that the model reacts consistently to changes in the structure.

#### 4.3.1.2. Discussion of Sources of Error

The high correlation between the calculated and measured data points demonstrates the ability of the model to determine the state of the structure. The requirements of specific applications, however, may require a smaller mean and standard deviation of error. For this reason, it is necessary to understand which facets of this model led to the errors.

One obvious source of error involves the system used to physically measure the coordinates. This source of error is not implicit in the model, but rather in the measurement system used to validate the model. The point locations were measured using polar coordinates and then converted to Cartesian coordinates. The device used to measure the radial coordinate is only accurate to one-sixteenth of an inch, and thus could contribute up to 0.15 cm to the error. Also, the device used to measure the angular coordinate was accurate to 0.5°. This could contribute an additional 0.8 cm to the error. Although the extent to which measurement error affected the results is not known, this source of error only affects the analysis of the system results and does not affect the precision of the sensing model.

Several sources of error do arise from inherent limitations of the sensing method. First, fiber optic sensing techniques exist which are more accurate than the intensity-based method used in this system. The air gap sensor is susceptible to noise caused by the mechanical coupling of the fiber to the beam, changes in the intensity of the light source, and electrical noise in the analog signals generated by the amplification and filtering system. Thus, some of the error observed for this system is undoubtedly caused by limitations of the sensing mechanism. This source of error could be reduced in future systems by replacing the intensity-based sensors with more sensitive, but higher cost, interferometric sensors.

Another source of error is that the calculations used in the system were formulated using a modified small deflection model. In many of the test configurations, the beams underwent large deflections. Although additional terms were added to the calculations to account for cases where large deflections occur, these terms were approximations and could not be expected to produce exact results. A more precise model could be obtain by using a complex large scale deflection model of the beams. Because this research focuses on the operation of the processing system, such an extensive structural examination is beyond the scope of this work.

### 4.3.2. Process Timing

Although processor speeds continue to increase, insight can be gained by examining the relative timing of processes for different algorithms. This section describes the relative process timing for three different algorithms and examines how the timing relationships change when math coprocessors are added to the processing nodes.

The basic elements of the synchronization sequence are the same for all of the algorithms. Figure 4.6 shows the synchronization sequence for the centralized processing algorithm using math coprocessors. The graph shows which processes are active in each processor, using the identifiers in Table 4.1, at time *t*. The processors are synchronized so that they sample their local sensors at time 0. The synchronization sequence commences when the BC receives a timer interrupt. It then broadcasts a synchronization command to the RTUs. The BC then goes into a wait state to account for the time that it takes the RTUs to get ready to sample their sensors. For the case shown in the figure, both of the RTUs are 8086 processors that operate at the same speed. Thus, the timing traces from these two machines are nearly identical.

For the case where split control and display are implemented, the 80286 RTU responds faster than the 8086 RTU. To account for this delay, the length of the wait loops in the BC and the 80286 RTU are adjusted so that all three machines sample their sensor at the same time. Even though the timing for this case is different than the case shown in the figure, the basic structure of the process, with the BC receiving the timer interrupt and then broadcasting a synchronization command to the RTUs, remains the same.

### 4.3.2.1. Processes Without Math Coprocessor

The timing diagrams for the centralized calculation, parallel calculation with combined control and display, and parallel calculation with split control and display algorithms executed by a system without math coprocessors are shown in Figures 4.7a, 4.7b, and 4.7c, respectively. For these algorithms, the sample period is 200 ms. The traces shown at the end of the sample period are actually the synchronization commands of the following sample period.

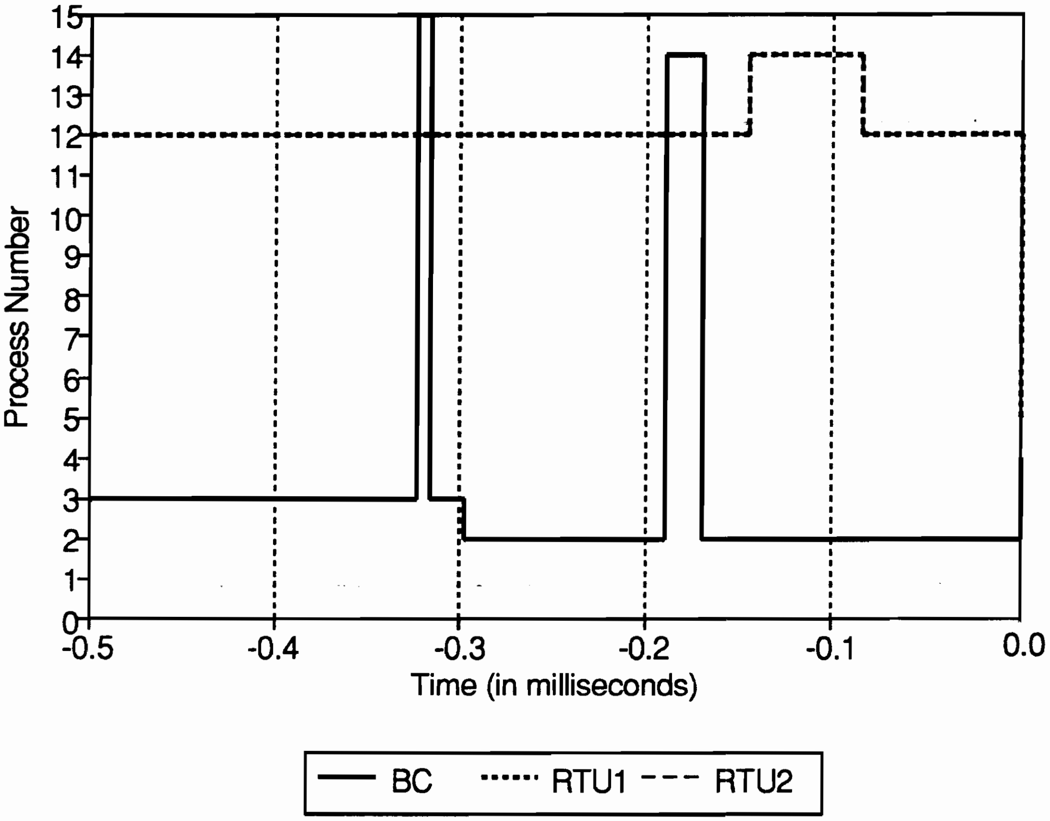


Figure 4.6. Synchronization sequence timing

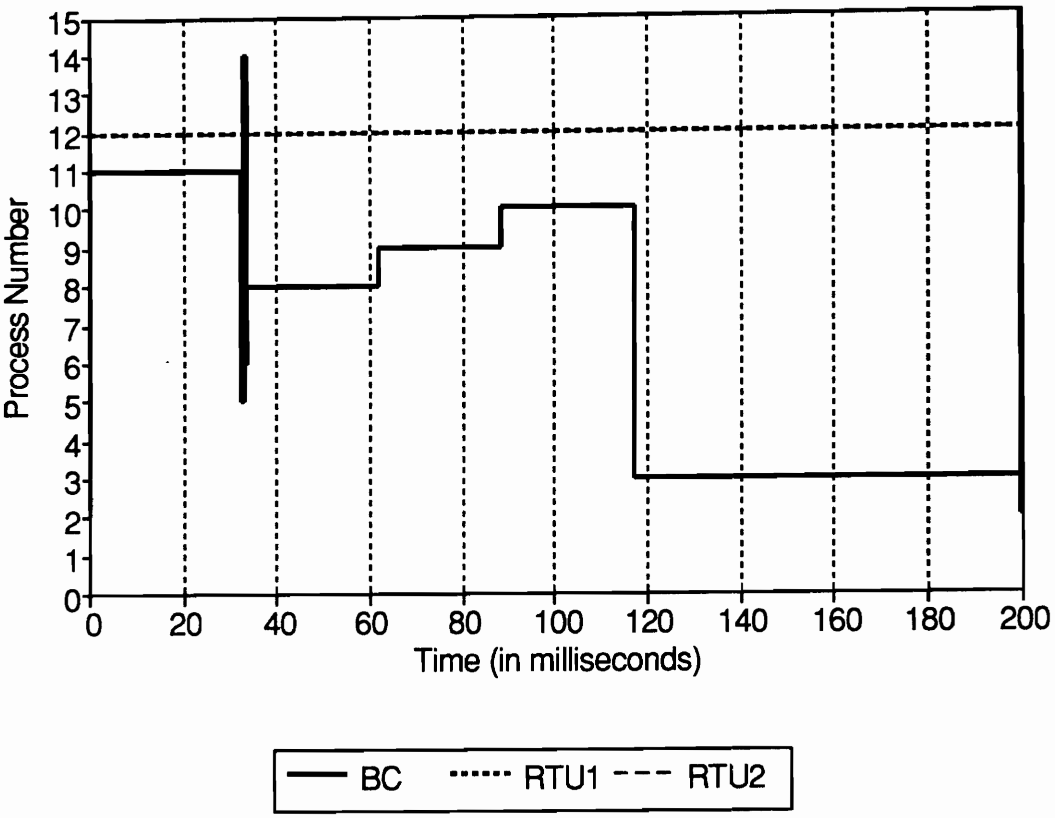


Figure 4.7a. Timing of centralized computation with combined control and display algorithm (without math coprocessing)

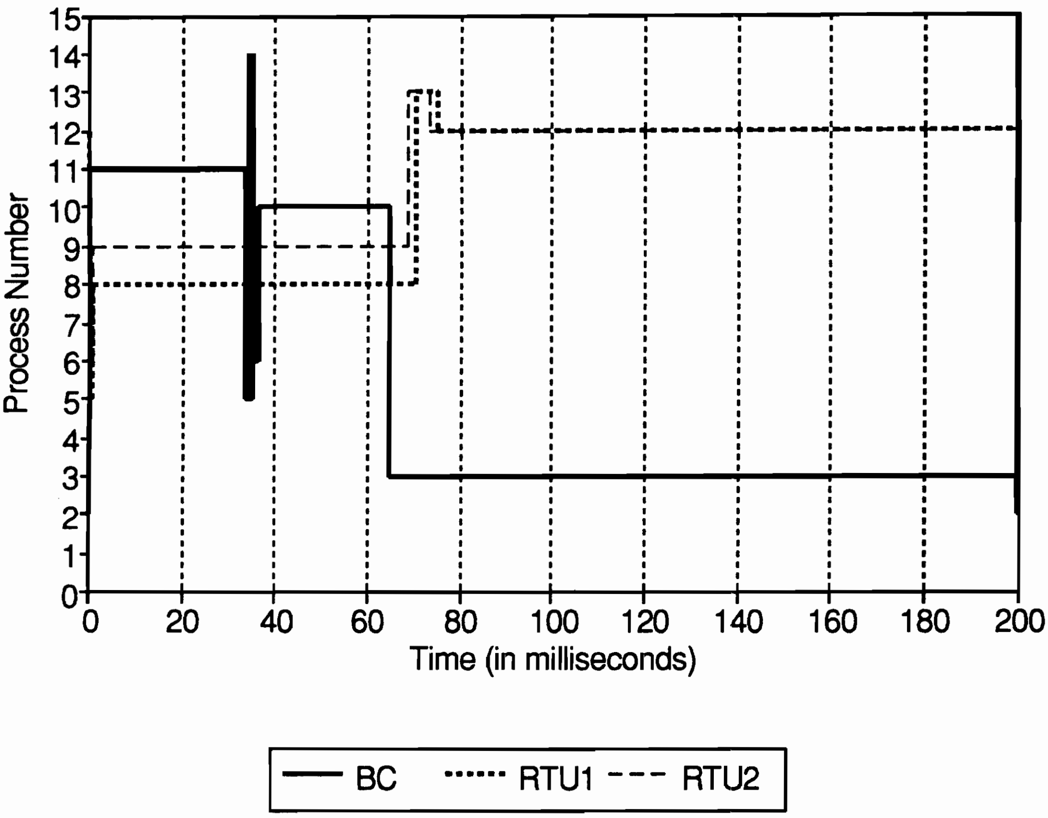


Figure 4.7b. Timing of parallel computation with combined control and display algorithm (without math coprocessing)

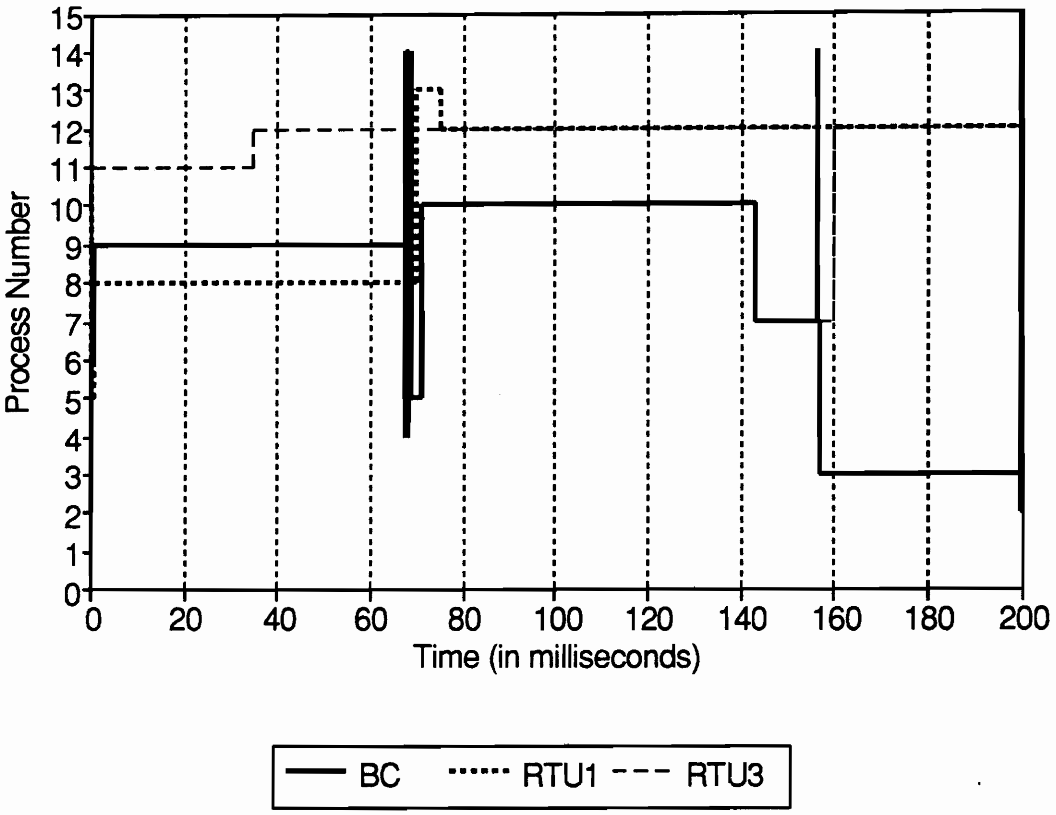


Figure 4.7c. Parallel computation with split control and display algorithm (without math coprocessing)

The process spike in the centralized calculation algorithm at 33 ms is shown in greater detail in Figure 4.8. At this time the BC is acquiring sensor data from the two RTUs. It is clearly shown that the RTU communication interrupts immediately follow the BC communication interrupts that occur when the BC requests sensor data from the RTUs. The process spikes shown in Figure 4.7b at 34 ms and in Figure 4.7c at 70 ms are essentially the same as the spike shown in Figure 4.8.

The single most important information shown in the timing diagrams is the amount of time that it takes a given algorithm to execute one sample period. The parallel calculation with combined control and display algorithm completes execution 75 ms into the sample period, while the centralized calculation and the parallel calculation with split control and display algorithms finish 115 ms and 160 ms into the sample period, respectively. Thus, the parallel calculation with combined control and display method has the fastest execution time of the three algorithms when math coprocessors are not used.

### 4.3.2.2. Processes With Math Coprocessor

The timing diagrams for the centralized calculation, parallel calculations with combined control and display, and parallel calculations with split control and display algorithms executed by a system with math coprocessors are shown in Figures 4.9a, 4.9b, and 4.9c, respectively. For these calculations, the sample period is 50 ms. There are several significant differences between these diagrams and those shown in Figure 4.7.

The centralized calculation algorithm remains basically the same in both cases. The only difference is that with the addition of math coprocessors, the time required to calculate the coordinates for each of the beams drops from 30 ms to 5 ms. This causes the execution time of one sample period to drop from 115 ms to 45 ms. Although this is a significant reduction in execution time, the centralized calculation algorithm, which is the second fastest algorithm when

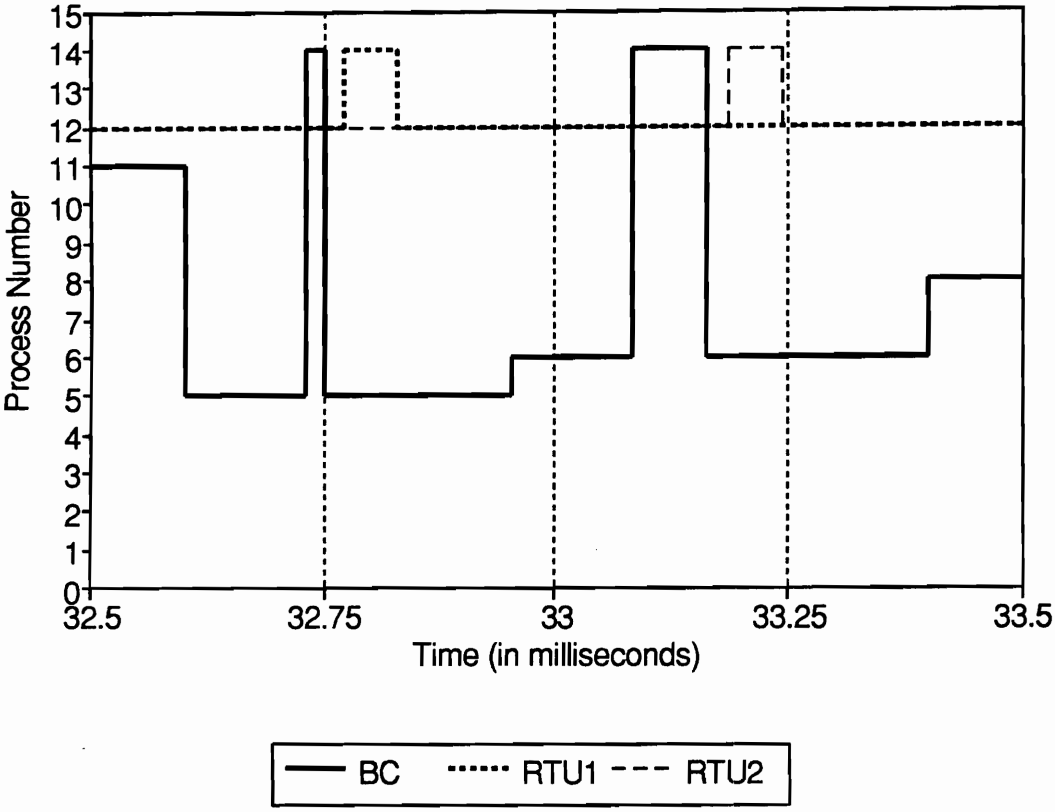


Figure 4.8. Processing spike in centralized computation algorithm without math coprocessing

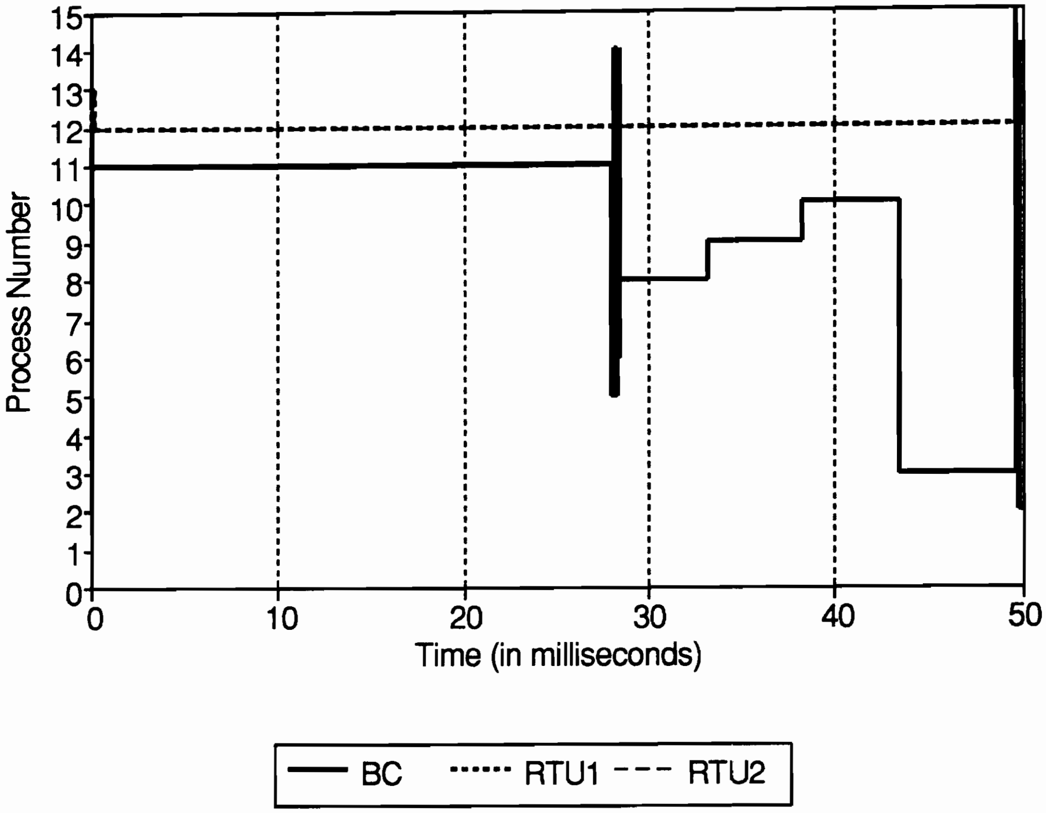


Figure 4.9a. Timing of centralized computation with combined control and display algorithm (with math coprocessing)

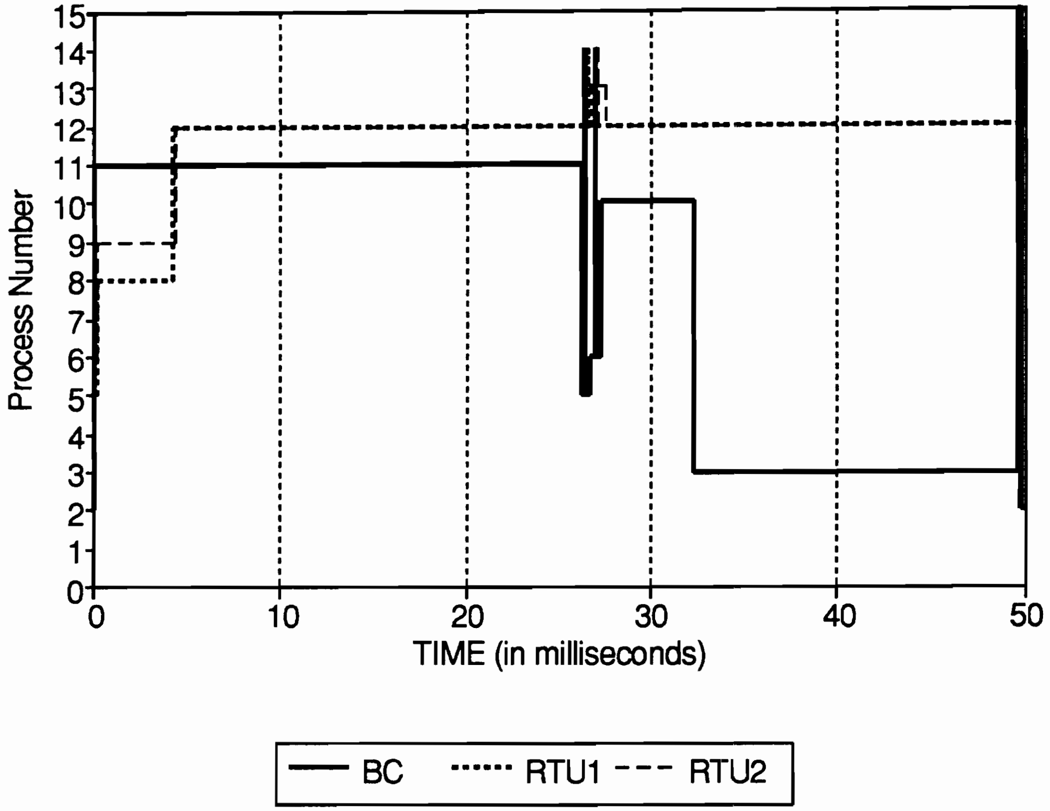


Figure 4.9b. Timing of parallel computation with combined control and display algorithm (with math coprocessing)

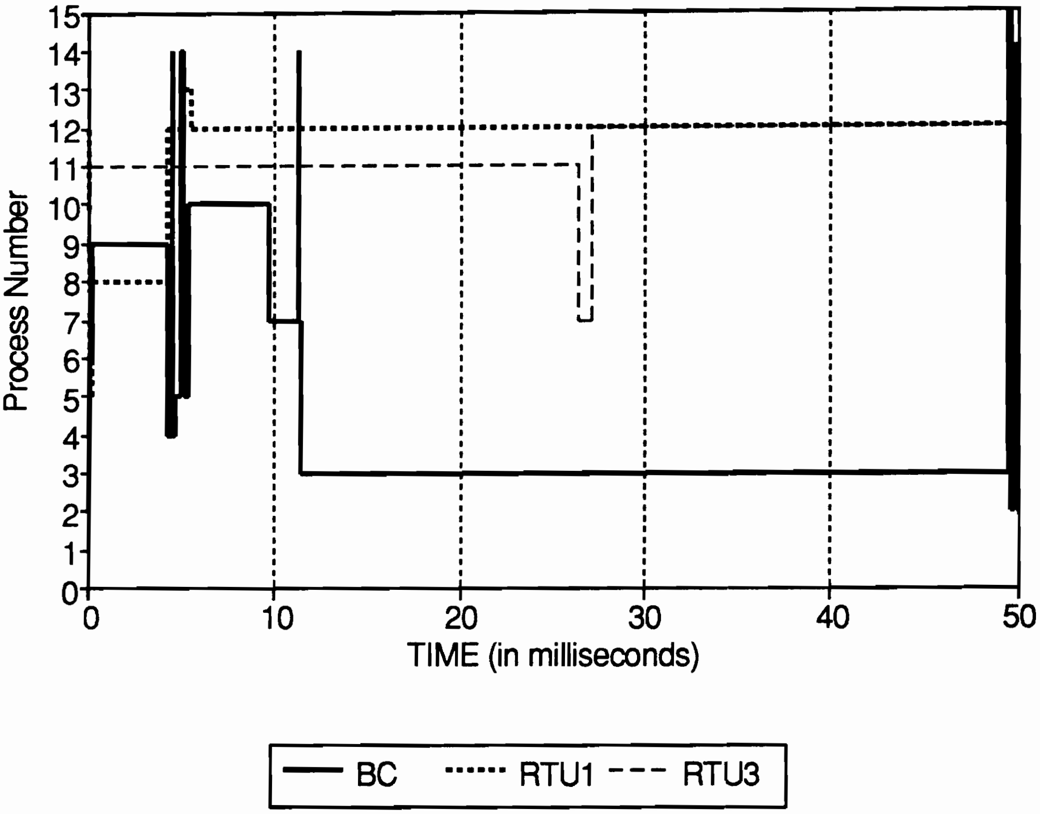


Figure 4.9c. Parallel computation with split control and display algorithm (with math coprocessing)

math coprocessors are not used, becomes the slowest of the three algorithms when math coprocessors are incorporated into the system. This is because the math coprocessor significantly influences the performance of the 80286 as it executes the centralized algorithm, but the operations performed on the 8086 machines are trivial and do not greatly benefit from the addition of the coprocessor. In the distributed algorithms, all three processors perform significant computations, and the benefits of the math coprocessors are useful in all three nodes.

The time required to execute one sample period using the parallel calculation with combined control and display algorithm drops from 75 ms to 32 ms when the system uses math coprocessors. This decrease in execution time is primarily due to the fact that the coordinate calculations that previously required 70 ms to complete on the 8086 machines take only 5 ms when math coprocessors are added to the system. Even though this processing time decreases by 65 ms, the processing time of the entire cycle only decreases by 43 ms. This is because the processing bottleneck in the system without math coprocessors occurs in the 8086 machines, but this bottleneck moves to the 80286 machine when math coprocessors are added to the system.

Another interesting comparison between Figures 4.7b and 4.9b is the location of the process where the RTUs save their beam coordinates, which has process identification code 13. In Figure 4.7b, when the RTUs finish calculating their beam coordinates, the BC has already read the coordinates that were calculated during the previous cycle. Thus, RTUs save their newly calculated coordinates immediately after the coordinate computations are complete. When math coprocessing is used, the RTUs complete their calculations before the BC asks for the coordinates calculated during the previous sample period. Thus, the RTUs wait for the BC to ask for the previous coordinate values before they proceed. The processing spike that occurs in Figure 4.9b at 27 ms is shown in greater detail in Figure 4.10. This figure shows that the RTUs save the coordinates that they calculated during the current sample period immediately after the BC reads the coordinates from the previous sample period.

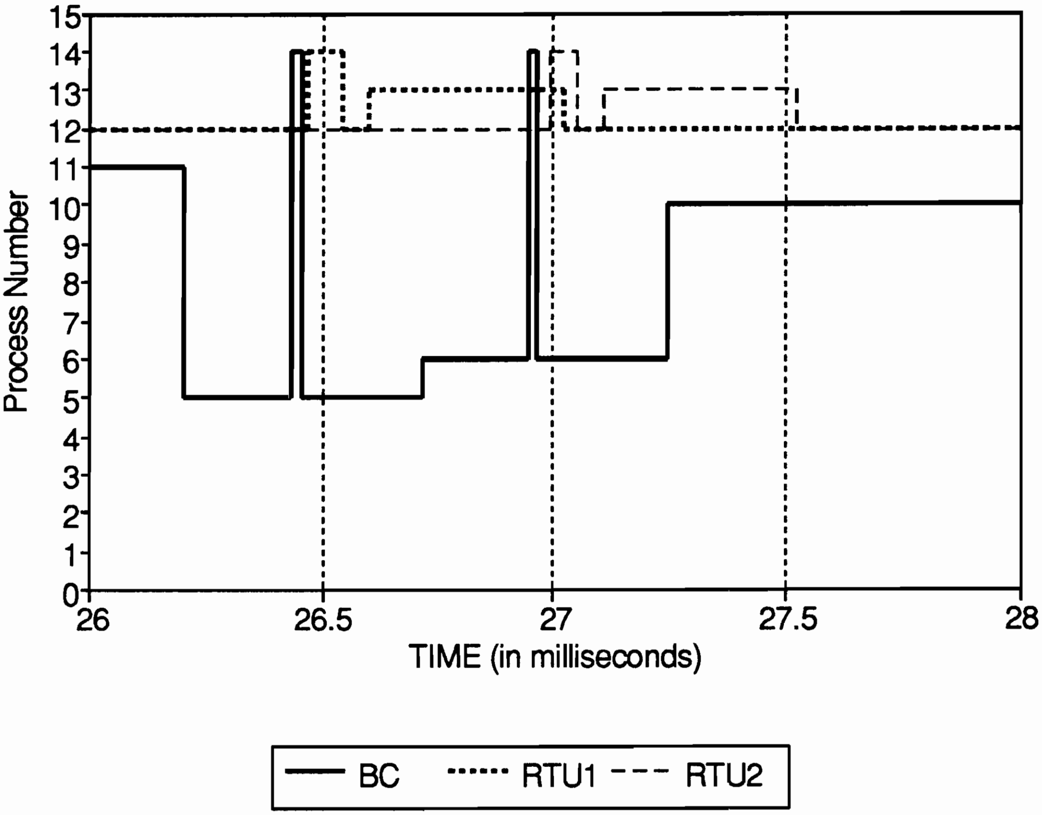


Figure 4.10. Processing spike in parallel computation algorithm with combined control and math coprocessing

The algorithm that gains the most advantage from the addition of the math coprocessors is the parallel computation with split control and display algorithm. This algorithm has the longest execution time when math coprocessors are not used, but it has the shortest execution time when math coprocessors are added to the system. When math coprocessors are not used, the 80286 machine finishes drawing the beam coordinates for the previous sample period at 35 ms, but it has to wait until the BC has completed its calculations before it can receive the beam coordinates calculated during the current sample period. The BC has to calculate coordinates for beams 2 and 3 in addition to receiving coordinates from beam 1 and sending the final coordinates to the 80286 machine. The BC is an 8086 machine, and it takes a long time to perform all of these tasks.

When math coprocessors are added, the execution times for the processes on both the 8086 BC and the 80286 RTU decrease. The processes executed by the BC, however, require more complex mathematical calculations than those performed by the 80286 machine. With the addition of math coprocessors, the BC can perform its calculations before the 80286 machine completes drawing the beam coordinates. Thus, the coordinates for the current shape of the structure are accessed by the 80286 machine immediately after it draws the shape of the structure for the previous iteration.

### *4.4. Summary*

The test system discussed in this chapter demonstrates the integration of functions that is required in smart structures. The test system shows that distributed processing can be used to determine the state of a structure. It also demonstrates, however, that the accuracy of the computations is directly related to the accuracy of the sensors and the mathematical model of the physical characteristics of the structure.

The relative timing of processes executed on separate processing nodes provides insight into the tradeoffs involved when processes are distributed among the processing elements. The experiments show that the addition of coprocessors affects the execution time and relative timing of processes and illustrate how these timing changes affect performance. Cases where the effects of data dependencies change as the execution times of the processes are altered are also demonstrated.

One attractive feature of distributed processing is that it can be scaled to fit the size of the structure. As structures increase in size, more computations are required to interpret the interaction between structural elements. With a distributed processing system, additional processors can be included in the system to handle this increased processing load. Increasing the number of processors, however, increases the complexity of process scheduling and the size of the communication network that connects the processing nodes. Additional work addresses these issues and attempts to determine how distributed processing can be effectively implemented in larger systems.

# Chapter 5. Multicomputer Implementation of a Damage DLE Algorithm

It is easy to construct and analyze structures with a small number of processors, but some envisioned smart structures require thousands of processors [19]. Results from small structures may not carry over to larger systems. This chapter extends the work in discussed in the previous chapter by presenting a distributed processing implementation of an algorithm that has been proposed for damage detection, location, and estimation (DLE) in a ten-bay planar truss. This implementation is meant to serve as a case study to provide insight into trends that may be encountered for distributed processing in large structures. The results of this study are expanded in later chapters to determine appropriate expectations for general systems that use distributed processing in large smart structures.

The chapter begins by describing a finite element model used to represent the dynamic physical response of the truss. A DLE algorithm proposed by Lindner, *et al.*, [51] is then discussed, and several different computer architectures are proposed for its implementation. These implementations use different interconnection network topologies and different methods for sharing data between the distributed processing nodes. This chapter concludes by presenting the measures that are used to evaluate the cost of each implementation.

### 5.1. Description of a Ten-Bay Planar Truss

The ten-bay truss, shown in Figure 5.1, is representative of truss structures that are often used in space applications [51]. Each bay of the truss consists of five structural elements, except for an incomplete eleventh bay that consists of only a single vertical element. The dynamic response of truss elements due to changes in loading and environmental conditions can be altered as the strength of elements in the truss change. Damage to truss elements can inhibit the range of functions that the truss is capable of performing. The degree and manner in which the truss capabilities are altered can be determined by examining the amount of damage in each truss element.

A finite element model (FEM) of the truss is developed in [51] to examine several truss parameters. The truss is assumed to be planar, and the location of each vertex in the truss is able to move in the plane with two degrees of freedom, one in the X direction and the other in the Y direction. The stiffness of the elements in the truss inhibits the motion of the vertices in the plane. The degree to which the motion of each vertex is limited is defined by the elasticity and placement of the truss elements. This elasticity is determined by the length and cross-sectional area of the truss elements as well as Young's Modulus of the material used to construct the truss. The number of degrees of freedom, denoted $n_h$, for a FEM of a truss is two times the number of vertices in the truss. For the ten-bay truss shown in Figure 5.1 there are 51 truss elements and 22 vertices, and thus 44 degrees of freedom.

The dynamic response of the truss can be determined from a stiffness matrix and a mass matrix defined for the truss. The stiffness matrix, K, is an $n_h$ x $n_h$ matrix that defines the connectivity of the truss elements. The mass matrix, M, defines the location of mass in the truss structure. With the $K_h$ and M matrices defined and negligible damping in the truss assumed, the dynamic mathematical model of the truss is

$$M\ddot{\eta}(t) + K_h\eta(t) = 0, \tag{5.1}$$

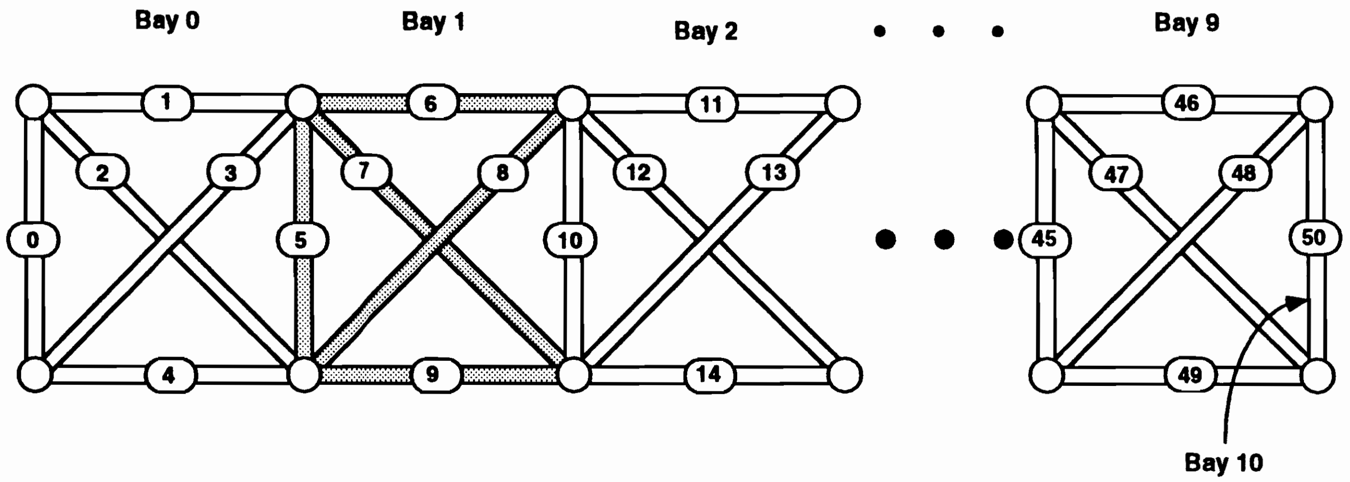


Figure 5.1. Ten-bay truss

where $\eta(t)$ is a vector of displacements for each degree of freedom in the model. The frequencies of vibration, denoted $\phi_j$, and the corresponding mode shapes for these frequencies, denoted $\omega_j$, can be determined by solving for the eigenvalues and eigenvectors of Equation 5.1. The mode shapes are given by a vector of $n_h$ elements. Each of the entries in the vector corresponds to the maximum amplitude of vibration for the degree of freedom corresponding to that entry. Thus, the mode shape vector defines the movement of each vertex in the truss for a given frequency of vibration.

In cases where catastrophic damage occurs, DLE algorithms are of little use because any functionality that the truss may have is destroyed. A more interesting case is where damage degrades the strength of certain elements in the truss, but leaves the truss structure basically intact. In such cases, the mass matrix, M, will remain the same but the stiffness matrix $K_h$ will change to account for a reduced stiffness in some of the truss elements. This change causes the frequency and shapes of vibration supported by the truss to change as well. Thus, damage in the truss can be detected by periodically measuring the shapes and frequencies of vibration in the truss.

### 5.2. The Damage Detection, Location, and Estimation Algorithm

The damage DLE algorithm proposed by Lindner, *et al.*, [51] is a multistep calculation that detects, locates, and quantifies the amount of damage in each truss element. A flowchart of the processes executed by the algorithm is shown in Figure 5.2. The algorithm begins by calculating the stiffness matrix and mass matrix for the truss. In addition, the algorithm calculates an element location matrix, $B_i$, for each of the truss elements. The $B_i$ matrix depends on the location of the *i*th element in the truss and the length, cross sectional area, and modulus of elasticity of the truss element. The $B_i$ matrix is defined in such a way that $B_i B_i^T$ is equal to the contribution of the *i*th structural element to the stiffness matrix. These initial

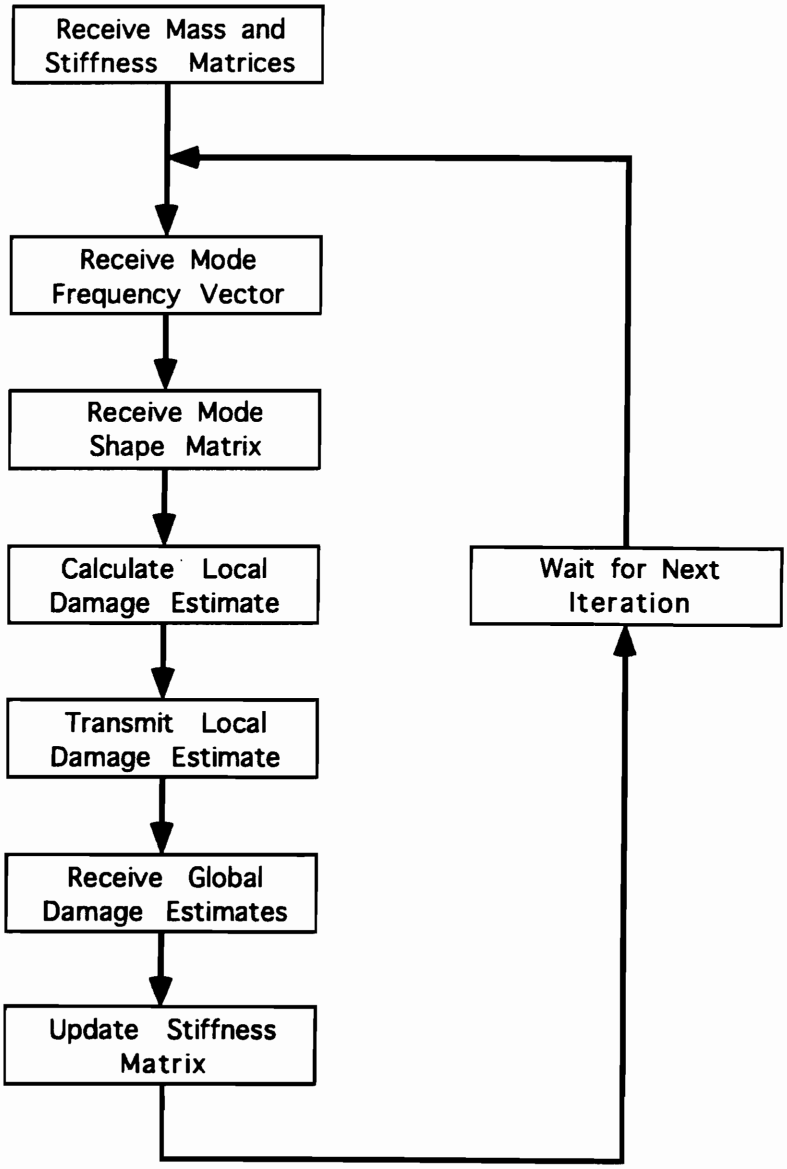

Receive Mass and Stiffness Matrices
Receive Mode Frequency Vector
Receive Mode Shape Matrix
Calculate Local Damage Estimate
Transmit Local Damage Estimate
Receive Global Damage Estimates
Update Stiffness Matrix
Wait for Next Iteration

Figure 5.2. Flowchart of the damage DLE algorithm

calculations for the stiffness, mass, and element location matrices need to be performed only at the start of the algorithm.

When damage occurs in the truss, the $B_iB_i^T$ contribution to the stiffness matrix is changed to $B_i(1-D_i)B_i^T$, where $D_i$ is a damage coefficient that represents the percentage reduction in strength of the *i*th truss element. If no damage has occurred to the *i*th truss element, $D_i$ is zero and $B_i(1-D_i)B_i^T$ reduces to $B_iB_i^T$. When the *ith* truss element is completely broken, $D_i$ is one and $B_i(1-D_i)B_i^T$ equals zero. Values for $D_i$ between zero and one represent cases where some damage has occurred to the element but it still provides some support.

At the beginning of each iteration, processed sensor measurements are used to determine the modal frequencies and corresponding mode shapes of vibration in the truss. Next, it is assumed that only one element of the truss has sustained damage. For each mode frequency-shape pair and truss element, the amount of damage that would have had to occur in that given truss element to cause the resultant mode frequency-shape pair is calculated. If the damage estimates in a given element are the same for all modes of vibration and have a value between 0 and 1, then that element is assumed to have sustained that amount of damage. If the damage estimates for a given structural element are widely scattered, then that element is assumed to not be damaged. After the damage to each element in the truss has been calculated, the stiffness matrix is updated so that future iterations will have knowledge of the elements that have been previously damaged.

The algorithm described above is just one example of a control algorithm that may be executed in a smart structure, but it exhibits many of the characteristics and types of calculations that are common in control functions. State vectors, matrix manipulation, and differential state equations are used in all classical control models [39]. Also, the examination of mode shapes and frequencies to determine the state of a structure is a widely used practice [50].

Thus, although this exact algorithm may not be executed in a particular system, it is representative of the types of algorithms that will be executed in smart structures.

### 5.3. Implementation

The algorithm proposed by Lindner, *et al.*, [51] defines the calculations required for damage DLE in a truss, but several different methods can be used to actually perform these calculations. Several features of the algorithm make it a suitable candidate for distributed processing. Specifically, the damage estimate calculation is performed for each truss element using information from the mode shapes and frequencies. The calculation for damage in one element during an iteration does not depend on the damage calculation for other elements. Thus, these calculations can be carried out in parallel. The damage calculation for a given element, however, does depend on the damage estimate for other truss elements during previous iterations. Also, the damage calculation for an element for one mode of vibration must be compared to damage estimates for that element and other modes.

From these observations concerning the algorithm, the calculation is decomposed as follows. A different processor is assigned the task of calculating the damage estimate for each element in the truss. Initially, the known mass and stiffness matrices for the healthy truss are transmitted to all processing elements in the structure. Also, each processor calculates the value of the location matrix, $B_i$, for the element to which it is assigned.

With this initial information in the processors, the iterative process of damage estimation can begin. At the beginning of each iteration, the mode shape array and mode shape vector are transmitted to each processor. Each processor uses this data to calculate the damage estimate for its assigned truss element. When the processor has computed its damage estimate, it transmits this estimate to all other processors in the system so that they can update their information on the current state of the structure. Each processor also receives the damage estimate for all other elements in the structure before the end of each iteration. If a processor does not receive information from some element in the structure, it should use the last damage estimate that is available from that element.

### 5.3.1. Topologies

After determining the information that must be shared between the processors, the data paths used to transmit this information must be defined. The topology of the interconnection network ultimately defines how data is shared among the processors. This work studies three different possible interconnection network topologies, the hypercube, the custom planar topology, and the custom hierarchical topology. The following sections define the location of communication links in each of the topologies, describe the potential benefits and drawbacks of each, and give reasons why each topology was chosen for this study.

#### 5.3.1.1. Hypercube

The first topology that is examined is a six-dimensional binary hypercube with 64 processing nodes. Each node in the structure is labeled with a six-bit binary address. The ten-bay planar truss contains 51 elements, but the number of nodes in a binary hypercube must be a power two. Thus, the implementation of a complete binary hypercube requires that some additional nodes be incorporated into the structure. These nodes are dummy processing nodes that will pass algorithm-based messages sent from other processing nodes, but do not generate any of their own algorithm-based traffic. For now, the existence of the dummy processing nodes is simply mentioned. The manner in which the dummy nodes are mapped onto the structure is discussed in Section 5.4.3.

As was discussed in Section 2.2.3.3, nodes that differ in exactly one bit location are connected with a communication link. These links are assumed to be embedded into the truss using the shortest possible physical path in the truss between the nodes. A message travelling between two nodes is routed by examining which bits in the address of the node that the message is currently at are different than those of the destination node. The message is routed to the link that corrects the highest order address bit of the current message location that is different than its corresponding address bit in the message destination. This process continues until the message reaches its destination.

The hypercube is examined in this study because of the many beneficial characteristics that it possesses that has made it a popular topology for general-purpose multicomputers. One beneficial characteristic is that the hypercube has a relatively small network diameter, and this helps to limit the number of nodes a message must pass through from its source to its destination. Also, the hypercube has regularity in its structure in that it has the same number of communication links at each node. Another interesting aspect of the hypercube is that it provides an example of a six-dimensional network being mapped onto a two-dimensional structure. It also allows the research to examine how the addition of dummy processing nodes that fill out the regularity of the structure affect the overall implementation.

#### 5.3.1.2. Custom Planar Topology

The second topology examined in this research is a custom planar topology. The motivation behind this topology is that the location of communication links is determined by the location of the physical truss elements. The location of the communication links can be understood by decomposing the truss into its individual bays. Each bay contains five truss local elements. Local elements 0 and 3 are the top and bottom horizontal elements respectively. Local elements 1 and 2 are the cross elements in the truss, and local element 4 is the vertical truss element in each bay. Figures 5.3a and 5.3b show how communication links are placed for each of the local bay elements. The location of communication links is the same in each of the

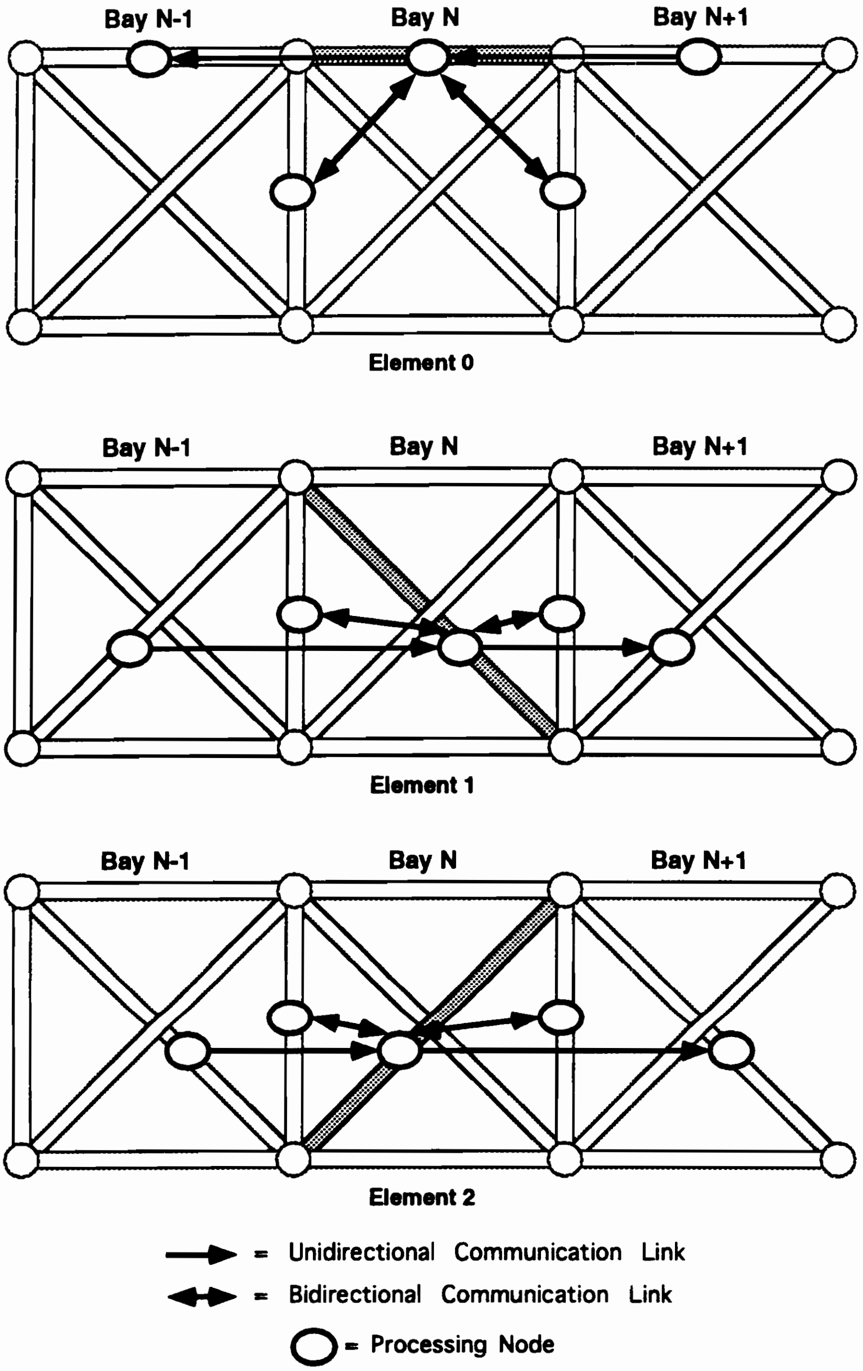


Figure 5.3a. Location of communication links in the custom planar topology for one truss bay

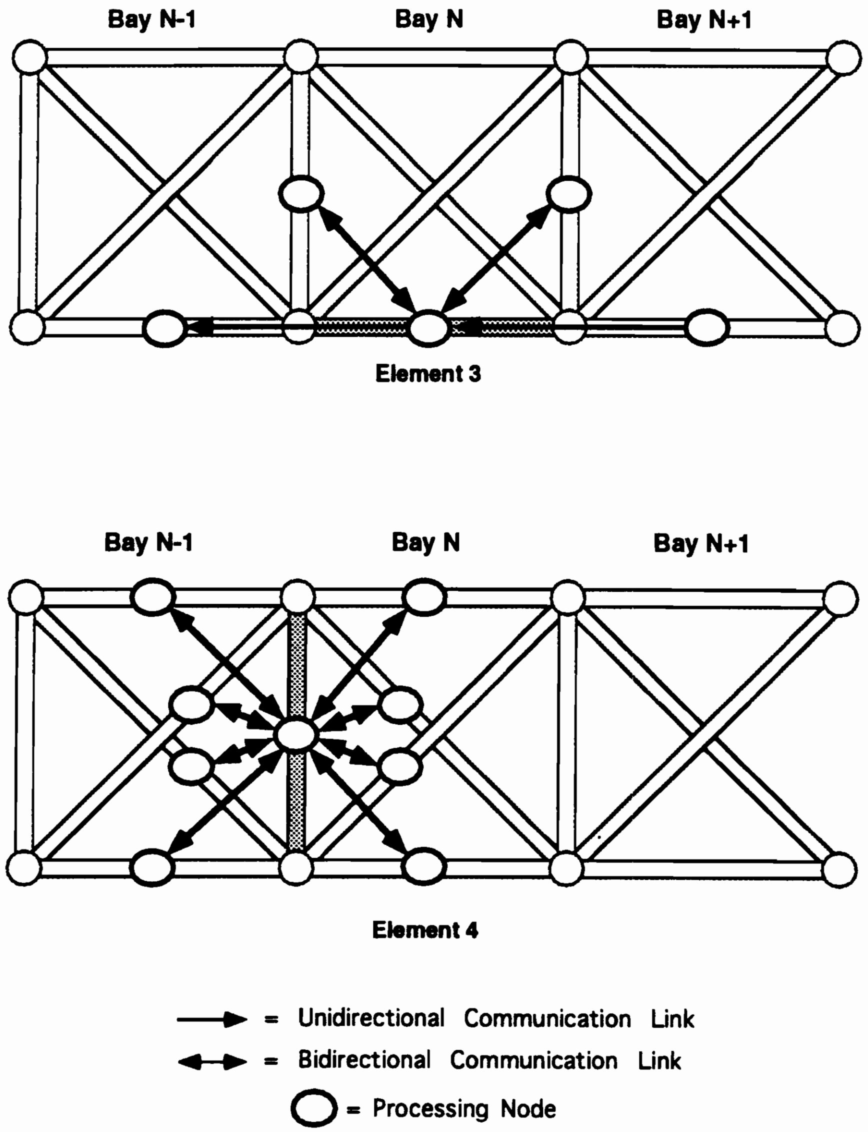


Figure 5.3b. Location of communication links in the custom planar topology for one truss bay

bays, with the obvious exception of the bays at the ends of the truss. Specifically, bay 0 contains no communication links that lead to lower numbered bays and bay 10 contains no communication links that lead to higher number bays. The arrows in the figure represent the existence and direction of the communication paths between the shaded truss element and the truss element containing the arrow.

As can be seen from the figure, not all truss elements contain the same number of communication links. Local element 4 is the only element of the bay that is connected to all of the elements in the bay. The other local bay elements can communicate with each other only by sending messages through element 4. Also, not all of the communication paths are bidirectional. Thus, the existence of a communication link from node J to node K does not necessarily imply the existence of a communication link from node K to node J.

Although the location of communication links in the custom planar topology are irregular, message routing can be accomplished according to the following rules. Local elements 0 and 3 are used to pass messages to lower numbered bays and local elements 1 and 2 are used to pass messages to higher bays. Thus, if a message is at a processing node in bay 6 and it has to go to a processing node in bay 2, it should be passed to either local element 0 or 3. Similarly, if a message is at a processing node in bay 3 and it has to go to a processing node in bay 8, it is passed to either local element 1 or 2. Messages are routed through bays until they reach the bay adjacent to their destination. At this point, the shortest route from the adjacent bay to the final destination is determined by examining which local element of the adjacent bay the message is at and which local element of the destination bay it needs to be passed to. If there is a link between two local elements, then the message is passed to this link. There is sufficient connectivity in the network to route a message in any local element of an adjacent bay to its destination in no more that two hops. If no direct link is available, one of the two-hop paths is chosen.

The custom planar topology represents an attempt to limit the amount of fiber that needs to be embedded in the structure. It provides data paths between any two processors in the network, but some messages may have to go through as many as eleven nodes before reaching their destination. This large network diameter is expected to cause some congestion in the wormhole routing mechanism. The large network diameter, however, may be counteracted by the degree of connectivity that exists between neighboring elements in the structure. The examination of this network helps to quantify how much message congestion occurs, and thus provides a measure that can be compared to its benefits concerning the limited amount of embedded fiber required for its implementation.

#### 5.3.1.3. Custom Hierarchical Topology

The custom hierarchical topology is a hybrid topology that attempts to obtain the small network diameter of the hypercube while limiting the amount of embedded fiber required in the structure and providing the degree of local connectivity that exists in the custom planar topology. In fact, the custom hierarchical topology contains a superset of the processing nodes and communication links that exist in the custom planar topology.

The custom hierarchical topology is a hierarchical architecture that is constructed by starting with a custom planar topology and then adding additional processing nodes and links. The custom planar topology forms the lower layer of the hierarchy. Four additional processing nodes, numbered 51 through 54, that are completely connected by bidirectional communication links, as shown in Figure 5.4, form the upper layer of the hierarchy. The custom planar topology contains eleven bays. This includes the highest numbered bay that consists of only a single vertical element. The processor embedded in local element four of each of the bays is connected to one of the nodes in the upper hierarchical layer by a bidirectional communication

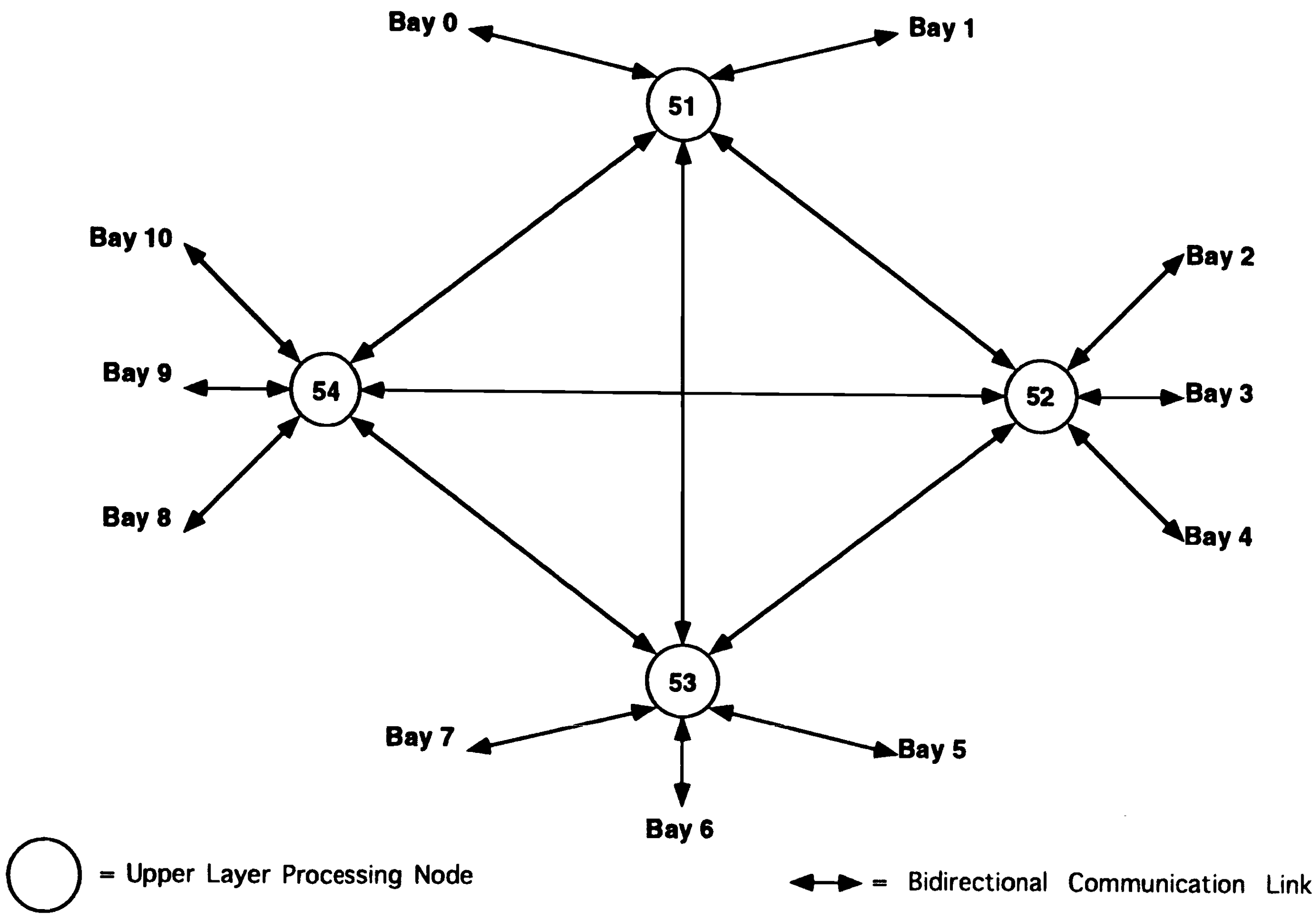


Figure 5.4. Upper level processing nodes in the custom hierarchical topology

link. Node 51 is connected to bays 0 and 1. Node 52 connects to bays 2, 3, and 4. Node 53 connects to bays 5, 6, and 7. Node 54 connects to bays 8, 9, and 10.

Since the upper layer of the hierarchy is completely connected, messages from one node in the upper layer to another node in the upper layer are simply transmitted over the direct link that connects the two nodes. Messages transmitted from a node in the lower layer of the hierarchy to a node in the upper layer of the hierarchy are first transmitted to local element four of the bay where the message originated, then passed to the upper layer of the hierarchy, then passed to the appropriate node in the upper layer. Messages transmitted from a node in the upper layer of the hierarchy to a node in the lower layer of the hierarchy are first transmitted to the node in the upper layer that is connected to the destination bay. The message is then transmitted to local element four of the destination bay and finally to the destination node.

Message routing from one node in the lower layer to another node in the lower layer is decomposed into two cases. If the source of the message is less than five bays from the message destination, then the message travels entirely through links in the lower layer of the hierarchy according to the routing rules that are used for the custom planar topology. If the source and destination are more than five bays away, then the message is passed to local element four of the source bay, then it is sent to the upper layer of the hierarchy. From this point on, the message follows the path discussed in the previous paragraph for messages whose source is in the upper layer and whose destination is in the lower layer.

The addition of the upper layer of the hierarchy reduces the network diameter to five. Also, the upper layer may significantly reduce congestion in the lower layer caused by messages that travel between distant bays by rerouting message that would otherwise have tied up a large number of links in the lower layer. Thus, not only do messages between distant bays arrive faster, but they also no longer interfere with communication between nearby processors. These benefits are gained at the cost of adding additional fiber to the custom planar topology. This study will quantitatively the improvement that is obtained with these additional links versus the amount of communication hardware that needs to be added.

### 5.3.2. Algorithm-Based Message Traffic

The topology of the interconnection network defines the location of communication links in the structure, and the location of these links determines the most efficient way for processors to share data to implement the algorithm. Two basic types of data transfer are required by the algorithm. First, broadcast communication must be available so that each processor receives mode frequency vector and mode shape matrix at the beginning of the iteration. At the end of the iteration, processors must have a way to send their damage estimate and receive the damage estimate computed by all other processors.

The use of wormhole routing makes it difficult for a single node to simultaneously broadcast a message to all other nodes in the system. This is because the paths to some nodes may be blocked while the paths to others remain clear. Thus, the transmitting node would have to wait until the message paths to all processing nodes become clear before transmitting. As an alternative to this method, messages flagged as broadcast messages propagate through the network in several hops. The exact method used to implement a broadcast depends on the topology used, but certain basic characteristics remain the same in each implementation. The source of a broadcast message sends the message to all of its neighbors. Nodes that receive a broadcast message also send copies of the message to any neighboring nodes that have not yet received the message. Using this method, the message eventually propagates to every node in the system.

When a node is transmitting its damage estimate to all other nodes in the structure, it does not use a broadcast transmission. This is because the communication links can be more effectively utilized if nodes first share and receive sensor measurements from local elements and then group the damage estimates from several local elements together and transmit this group of estimates to more distant portions of the structure. The method used to broadcast and share of damage estimates is discussed in the following sections.

### 5.3.2.1. Hypercube

The regular structure of the hypercube makes it possible transmit broadcast messages in a straight-forward manner. The source of a broadcast message transmits copies of the message along its six outgoing communication links and sets a flag in the header of each of these transmissions that informs the receiving node that it is a broadcast transmission. The communication links entering a node are numbered 0 through 5, where link $j$ connects the node to the node in the system that has the same binary address except bit $j$ is inverted. Each node that receives a broadcast message examines which link the message was received on. In addition to sending the message to its processing element, the receiving node also broadcasts copies of the message to all of the outgoing links that have a lower number than the link that the message was received on. For example, if a broadcast message is received on link 4, it is transmitted on links 0, 1, 2, and 3. Using this method, a broadcast message will propagate to all nodes in the network, and only one copy of the message will arrive at each node.

Grouping and transmitting damage estimates can also be easily performed on the hypercube. After an element computes its damage estimate, it transmits this value on outgoing communication link 0. In turn, it waits to receive the damage estimate of the neighbor on incoming communication link 0. The node then groups its damage estimate with the estimate it received from its neighbor and transmits this value over outgoing communication link 1. The node waits to receive a group of two damage estimates on incoming communication link 1. At step $j$ in this process, the nodes receive $2^{j-1}$ damage estimates. In turn, they are able to transmit the $2^{j-1}$ values they already had plus the additional $2^{j-1}$ values they just received, which makes a total of $2^j$ values, on link *j*. In this way, by the time the node receives data on the highest order link, it knows the damage estimate of every element in the system.

### 5.3.2.2. Custom Planar Topology

The irregular structure of the custom planar topology makes broadcast transmission more difficult. The damage DLE algorithm uses broadcast transmission only to send the mode frequency vector and mode shape matrix. It is assumed that these matrices are available at the beginning of each iteration in node 0 at the far left side of the truss. Node 0 has the responsibility of transmitting these matrices to all other processing nodes.

The broadcast method is designed so that once local element 4 of a given bay receives a broadcast, it sends copies of the message to all of the processing nodes in its bay. Thus, the broadcast problem reduces to the problem of how to send messages to local element 4 in each of the bays. This problem is solved by predefining which bays the local element 4 processors should transmit their broadcast messages to. The following definition is used for broadcast message transmission. Bay 0 transmits to bay 1 and bay 6. Bay 1 transmits to bays 2 and 4. Bays 2 and 4 transmit to bays 1 and 3 respectively. Bay 6 transmits to bays 7 and 9. Bays 7 and 9 transmit to bays 8 and 10 respectively. Bays 3, 5, 8, and 10 do not transmit broadcast messages to any other bays. Using this definition, the modal frequency vector and mode shape matrices can be successfully transmitted to all of the processing nodes.

Sharing the damage estimates between all elements in the structure takes place in four steps. First, all of the elements in a bay transmit their damage estimate to local element 4 of that bay. Thus, local element 4 contains information about all elements in its bay. The next two steps of the algorithm take place simultaneously. Local element 4 of bay 10 transmits its group of damage estimates to local element 4 of bay 9, while local element 4 of bay 0 transmits its group of damage estimates to local element 4 of bay one. When local element 4 in bay *j* receives a group of damage estimates from bay *j-1*, it appends damage estimates from its own bay to this data and passes it to bay *j+1*. Similarly, when local element 4 in bay *j* receives a group of damage estimates from bay *j+1*, it appends damage estimates from its own bay to this data and passes it to bay *j-1*. When local element 4 of a given bay has received and transmitted data to both the upper and lower neighboring bays, then it has received the damage estimates from all other nodes in the structure. It transmits this data to all of the other elements in its bay. The cycle is complete when all of the local element 4 processors have transmitted their data to all of the local elements in their bay. At this point, all processors in the structure have received the updated damage estimates for all elements in the structure.

#### 5.3.2.3. Custom Hierarchical Topology

Message broadcasts in the custom hierarchical topology are similar to broadcasts in the hypercube. The source of the broadcast transmits the message to every link that it is attached to. Every node that receives a broadcast message examines which incoming link the message was received on and determines which outgoing links, if any, the message should be transmitted on.

The following rules are used to determine where to transmit messages. Only upper layer processors and local element 4 processors in the lower layer retransmit broadcast messages they receive. If a local element 4 processor receives a broadcast from a node in the upper layer of the hierarchy, the message is transmitted to all other local elements in the bay. If a local element 4 processor receives a broadcast message from some node in its own bay, it transmits the message to all of the other local elements in its bay as well as to the node in the upper layer that it is attached to. When a node in the upper layer of the hierarchy receives a broadcast from some node in the lower layer, it transmits the message to every node it is attached to except the node that originally sent the message. Finally, if a node in the upper layer receives a broadcast from another node in the upper layer, it transmits the message to all of the lower layer nodes that it is attached to. Using these rules, any node can generate a broadcast message and have a single copy of the message arrive at every node in the structure.

Damage estimates are shared among the processing nodes in the custom hierarchical topology using a mixture of the techniques used for the hypercube and the custom planar topology. All of the local elements in a bay transmit their damage estimate to their local element 4 node. After the local element 4 node receives damage estimates for all elements in the bay, it transmits the group of estimates for the bay to the processor in the upper layer of the hierarchy to which it is attached. The upper layer processor, in turn, waits to receive damage estimates from all of the bays that it is attached to. After receiving these estimates, the upper layer processor groups this data and transmits it to all other processors in the upper layer. Once an upper layer processor receives the damage estimates from all of the other processors in the upper layer, it has a complete knowledge of all of the damage estimates in the structure. At this point, the upper layer processor transmits the damage estimates back to all of the lower layer local element 4 processors that it is attached to. The local element 4 processors complete the algorithm by sending these damage estimates to all of the elements in their bay. The cycle is complete when all processors in the system have received a complete set of damage estimates for the current iteration and update their stiffness matrix to account for any changes that have occurred.

### 5.3.3. Background Message Traffic

In addition to the traffic generated by the damage DLE algorithm, there may be other message traffic that needs to be transmitted in the network. It is important for the network to be able to handle background message traffic while still carrying the message traffic generated by the damage DLE algorithm.

The affect of background message traffic depends on the amount and distribution of background traffic generated in the network. In this study, both the length of background messages and the time between successive background transmissions are assumed to be exponentially distributed. Exponentially distributed generation times are appropriate for message traffic that may be generated by a number of independent processes [116]. The mean message length remains constant. This is because different processes may tailor their message size to some length that can be efficiently handled by the network. The amount of traffic in the network is varied by changing the mean time between successive transmissions. By varying this parameter, it is possible to examine how the performance of the damage DLE algorithm is affected by network loading.

It should be noted that the three different topologies studied in this research use different numbers of processors. The hypercube uses 64 processors, the custom hierarchical topology uses 55, and the custom planar topology uses 51. This model assumes that the background message traffic generation rate is specified on a per node basis. Thus, the 64 nodes in the hypercube produce more background traffic than the 51 nodes of the custom planar topology if the average generation rates per node are equivalent for the two networks. This is reasonable because it may be assumed that the additional processors that were added to complete the hypercube would be assigned extra computational tasks. These tasks would require additional background message traffic when they are executed. Also, the addition of extra processors could also cause background processes to have finer granularity, and thus require more interprocessor communication.

## *5.4. Cost Metrics*

The performance of a multicomputer network can be enhanced by adding processors and communication links in appropriate locations. These additional elements increase the system cost. The performance of a topology must be measured against this cost. The cost of the topologies used in this study are judged by examining the number of communication links in the topology, the number of processing nodes, and the amount of fiber that is embedded in the structure. Table 5.1 summarizes costs for each topology. The following sections discuss details of each cost calculation. The change in cost as the size of the structure increases is also examined.

### 5.4.1. Number of Communication Links

The number of communication links in a network gives a measure of the amount of support hardware, such as line drivers and message buffers, that must be implemented at each node [76]. The excellent performance of the hypercube for a wide range of problems can be attributed to its rich connectivity, but this connectivity is gained at the cost of requiring 384 communication links for a 64-node hypercube. The custom planar topology, on the other hand, requires only 196 communication links for a 51-node network. The custom hierarchical topology requires an additional 34 communication links than the custom planar topology, for a total of 230 links for a network with 55 processors.

### 5.4.2. Number of Processors

The next cost metric is the number of processors in the network. One processor is assigned to each element in the structure, so a minimum of 51 processors are required for the 51 structural elements in the ten-bay truss. The custom planar topology requires just those 51 processors. The custom hierarchical topology requires four additional processors to form the upper layer of the hierarchy over the custom planar topology, for a total of 55 processors. The hypercube must have a number of processors that is a power of two and thus requires 64 processors.

Table 5.1. Cost summary

| Topology | Hypercube | Planar | Hierarchical |
|---|---|---|---|
| Length of Embedded Fiber | 340 to 931 | 220 | 298 |
| Number of Communication Links | 384 | 196 | 230 |
| Number of Processors | 64 | 51 | 55 |

The processors added to the custom hierarchical and hypercube topologies do not perform any of the algorithm-based computations. The communication controllers in these processing nodes route algorithm-based messages through the network, but the processors do not generate any of their own algorithm-based messages. Thus, these additional processor are beneficial in that they help to facilitate message routing, but their full worth to the network as far as performing processing tasks unrelated to the damage DLE algorithm is not considered. The cost of including them in the topology is considered.

### 5.4.3. Amount of Embedded Fiber

In addition to the hypercube requiring additional communication links, the length of the links must also be considered. Embedded fiber in a composite material increases the manufacturing cost and complexity [28, 112]. An added degree of complexity is created at the vertices of the structure where the communication links of one element are connected to the next element. The amount of fiber embedded in the structure is a measure of the complexity required to manufacture the structure.

The length of the required embedded fiber is normalized to the length of a single horizontal strut, so one unit of length is equal to the length of one horizontal strut in the structure. The bays are square, so each vertical strut also has a length of 1 and each diagonal strut is of length 1.414. It is assumed that the processing nodes are embedded in the center of the strut.

For the custom planar topology, each processor is connected only to processors in neighboring struts. Thus, there are only three possible communication link lengths. A link from a horizontal or vertical element to another horizontal or vertical element, denoted as a type A link, has length 1. A link from a diagonal element to a neighboring diagonal element is a type B link and has length 1.414. Finally, a link from a diagonal element to a horizontal or vertical element is a type C link and has length 1.207. The amount of embedded fiber is calculated by summing the amount of embedded fiber in each bay. The length of embedded fiber for a link is attributed to the bay that contains the element that drives the communication link, not for the bay that receives the transmission. This prevents counting links twice.

Examining the length of the communication links shown in Figures 5.3a and 5.3b for each bay yields the following results. Bay 0 requires six type A links, two type B links, and six type C links. Bays 1 through 8 require ten type A links, two type B links, and eight type C links. Bay 9 requires ten type A links, zero type B links, and eight type C links. Bay 10, which consists of a single vertical element, requires two type A links, zero type B links, and two type C links. By summing all of the link types, it can be seen that the entire network requires a total of 98 type A links, 18 type B links, and 80 type C links. This means that 220 units of fiber are required in the custom planar topology.

The custom hierarchical topology contains the same links that exist in the custom planar. In addition, the links of the upper layer of the hierarchy also have to be embedded in the structure. Each node in the upper layer attach to several nodes in the lower layer of the hierarchy. The upper layer nodes are embedded near the lower layer nodes that they are attached to. Node 51 is embedded in element 5, node 52 in element 15, node 53 in element 30, and node 54 in element 45. In this configuration, the upper layer links use 79 units of embedded fiber, and the entire custom hierarchical topology requires a total of 298 units of embedded fiber. There may be other possible node placements that slightly reduce the required amount of embedded fiber, but the placement used in this implementation is modular and can be easily expanded if a larger truss is used.

The mapping of a six dimensional hypercube onto a two-dimensional ten-bay truss presents several interesting problems. First, it is not readily apparent where to place the extra hypercube nodes that do not perform calculations for a specific structural element. Also, because each node has neighbors in six different dimensions, it is difficult to determine efficient relative placements for these neighbors.

The problem of mapping of an algorithm onto a topology is much like the problem of mapping an interconnection network topology onto a structure. In the first case, the designer attempts to map processes to processing nodes in a manner that limits the distance between all pairs of processors that are required to share data. In the latter case, the designer attempts to map processing nodes to the elements of a structure in a manner that limits the distance between processors that are neighbors in the topology. The problem of determining an optimal mapping of an arbitrary problem onto a general topology has been shown to be NP-complete [117]. Thus, it is unlikely that a efficient algorithm for mapping a general interconnection network topology onto a physical structure will be found.

One method that could theoretically be used to determine the most effective mapping is to perform an exhaustive search of all of the possible mappings. Unfortunately, the complexity of such a computation is on the order of (N-1)!, where N is the number of processors. Clearly, this method is too computationally demanding to be solved for a 64-node hypercube.

Because it is impractical to determine the optimal mapping to limit the amount of embedded fiber, this research attempts to determine upper and lower bounds for the amount of fiber that is required. A lower bound is determined by using a modular technique. First, the number of truss elements in a single bay is reduced from five to four, as is shown in Figure 5.5. Next, the number of bays in the truss is reduced from 10 to 8. Since the remaining eight-bay truss is a subset of the elements in the ten-bay truss, it will clearly require less embedded fiber to implement.

To find a lower bound for the required amount of embedded fiber, this research examines the minimum amount of fiber required for each dimension of the hypercube. First, a single truss bay is examined. Two address bits can be used to determine the location of these four elements

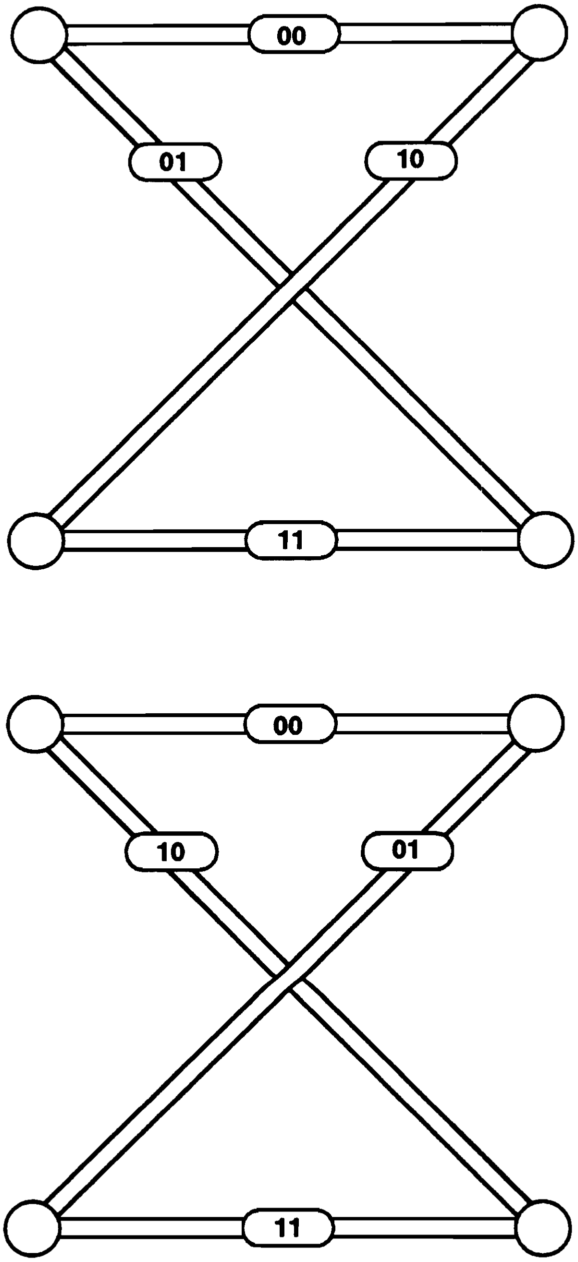


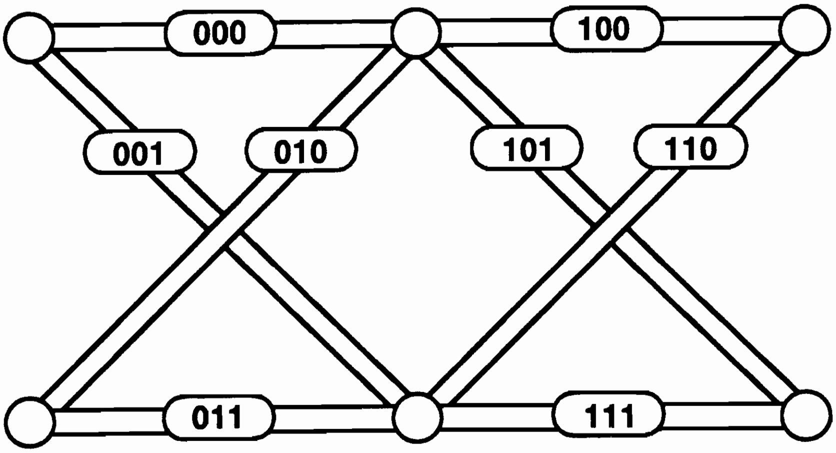
Figure 5.5. Hypercube implementations of a four-element bay

in a bay. Since there are only four elements, an exhaustive search of all processor placements is possible and reveals that the two processor placements shown in Figure 5.5 are optimal and require 9.65 units of fiber.

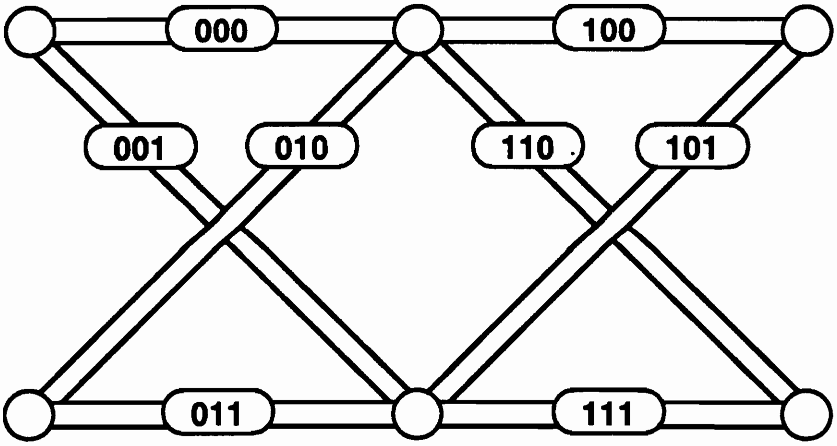
After determining the proper arrangement of the two low order address bits in one bay, it is necessary to determine the best way to arrange the address bits of neighboring bays. Figure 5.6 shows two possible methods for mapping processors onto a two-bay truss. Mapping A requires 33 units of embedded fiber, but mapping B requires only 29 units. From this description, it is seen that the mapping of processing nodes to the diagonal elements of the bay should alternate every other bay to reduce the amount of embedded fiber for links in the diagonal elements of neighboring bays.

Three upper address bits are used to designate which of the 8 bays a given element belongs to. Thus, all elements in the same bay have the same upper three address bits. Because of the isomorphic nature of the problem, the elements in bay 0 are assigned the address 000XX, where the lower two-bits (XX) determine which element of the bay is represented. If we assume that the placement of the lower address bits is optimal, then the optimal solution for the entire structure can be calculated by exhaustively searching 7! possible ways to arrange the remaining 7 bays. This search found that such a truss implementation requires 244 units of embedded fiber to make all connections between bays. In addition, each of the eight bays requires 9.65 units of fiber for the connections internal to the bay. Thus, the eight-bay truss with four elements per bay requires a minimum of 321 units of embedded fiber.

At least 9.65 units of fiber are required for the internal connections of the 9th and 10th bays in the ten-bay truss. Thus, the ten-bay truss requires at least 340 units of embedded fiber. Several factors used to calculate this lower bound should be examined. First, only 40 of the 64 processors are mapped onto the structure. This means that the length of the six links attached to the 24 unused processors is not considered. Also, the bays in the ten-bay truss require a sixth



a. Non-optimal arrangement



b. Optimal arrangement

Figure 5.6. Arrangements of neighboring bay elements

address bit because there are ten bays instead of eight. Finally, the vertical element in each truss bay is not considered. For these reasons, it is expected that the actual amount of embedded fiber required is much larger than this lower bound.

An upper bound for the optimal amount of embedded fiber required to map a hypercube onto a ten-bay truss can be determined by calculating the amount of fiber required for a specific implementation. Table 5.2 shows the mapping of processors onto the elements of the structure. This mapping was created by starting with the minimum mapping described for an eight-bay truss with four elements per bay, and then adding necessary processing elements and links that are required for a ten-bay truss with five elements per bay. An attempt was made to minimize the amount of embedded fiber when additional processors and links were added, but it is not known whether this mapping is optimal. Figure 5.7 shows the connections that are required for one of the processors in the structure. From this figure, it can be seen that although some of the links connect neighboring elements in the structure, other links require longer connections. It should also be noted that more than one processor is mapped to some of the elements. This accounts for the processors that were added to make the total number of processors in the structure a power of 2. The amount of fiber needed for this implementation is 931 units.

The upper bound is much greater than the lower bound of 340 units, but it must be remembered that this lower bound was calculated by optimizing the mapping for some of the links and not considering the mapping of remaining links. Since the upper bound was calculated by considering the placement of all links in the structure, and because an attempt was made to make this mapping optimal, it is probable that the upper bound is closer to the true amount of embedded fiber required to implement the six-dimensional hypercube than the lower bound.

Table 5.2. Mapping of processing nodes to truss elements

| Processing Node Number | Truss Element Number | Processing Node Number | Truss Element Number |
|---|---|---|---|
| 0 | 1 | 32 | 35 |
| 1 | 2 | 33 | 30 |
| 2 | 3 | 34 | 20 |
| 3 | 4 | 35 | 25 |
| 4 | 6 | 36 | 0 |
| 5 | 8 | 37 | 5 |
| 6 | 7 | 38 | 15 |
| 7 | 9 | 39 | 10 |
| 8 | 16 | 40 | 40 |
| 9 | 18 | 41 | 45 |
| 10 | 17 | 42 | 35 |
| 11 | 19 | 43 | 50 |
| 12 | 11 | 44 | 35 |
| 13 | 12 | 45 | 35 |
| 14 | 13 | 46 | 35 |
| 15 | 14 | 47 | 35 |
| 16 | 36 | 48 | 41 |
| 17 | 38 | 49 | 42 |
| 18 | 37 | 50 | 43 |
| 19 | 39 | 51 | 44 |
| 20 | 31 | 52 | 46 |
| 21 | 32 | 53 | 48 |
| 22 | 33 | 54 | 47 |
| 23 | 34 | 55 | 49 |
| 24 | 21 | 56 | 35 |
| 25 | 22 | 57 | 35 |
| 26 | 23 | 58 | 35 |
| 27 | 24 | 59 | 35 |
| 28 | 26 | 60 | 35 |
| 29 | 27 | 61 | 35 |
| 30 | 28 | 62 | 35 |
| 31 | 29 | 63 | 35 |

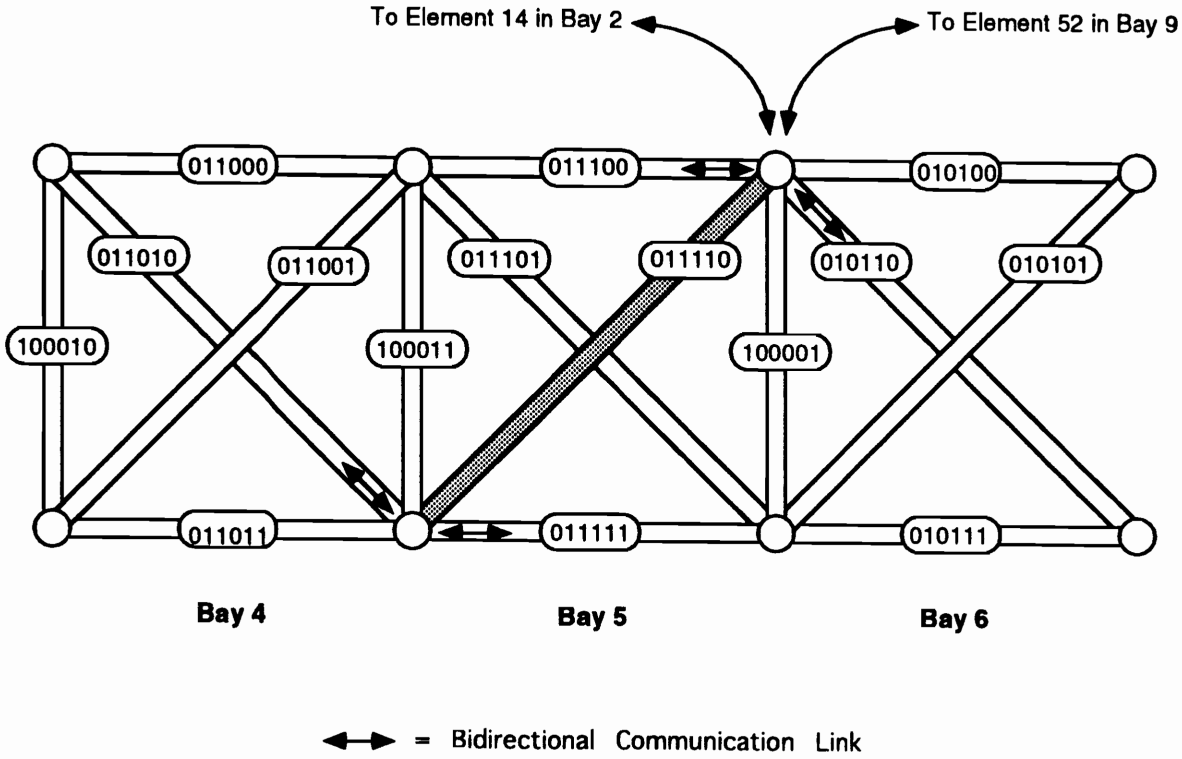


Figure 5.7. Sample hypercube connections for one truss element

### 5.4.4. Scalability

The measures computed in the previous section measure the costs associated with the implementation of each topology in the ten-bay truss. The choice of a ten-bay truss was somewhat arbitrary, so it is important to examine how the cost of the system changes as the size of the truss increases. The following paragraphs discuss how each of the cost functions scales as the size of the truss is modified.

Each bay requires a single processor for each of the five elements in the bay. In the custom planar topology, one processor is required for each element in the truss, so the number of processors is equal to the number of elements in the truss. The custom hierarchical topology requires processors in upper layers of the hierarchy in addition to the processors already contained in the custom planar topology. The arrangement of nodes in the upper layer of the hierarchy may change as the size of the truss increases, and this arrangement would have to be specified before an exact equation for the total number of processors in the system could be formulated. In all possible arrangements, however, a single higher layer processor oversees a group of lower layer processors. Thus, the lower layer of the hierarchy contains a majority of the processing nodes. Since the number of processors in the lower layer is equal to the number of elements in the truss, and because this term dominates the total number of processors required, the number of processors in the custom hierarchical topology scales linearly with the size of the truss.

For hypercube implementations, the number of truss elements should ideally be a power of two. In these cases, the number of processors is exactly equal to the number of truss elements. In reality, up to (N-2) additional processors, where N is the number of truss elements, may have to be added to the structure to make the total number of processors in the hypercube a power of two. Thus, although the number of processors required scales linearly, it does not scale uniformly. Instead, the number of processors is a step function that remains steady as the number of truss elements is increased until this number exceeds a power of two, at which time the number of required processors doubles. Thus, a truss with *N* elements requires $2^{int(\log_2 N)+1}$ processors, where *int* is the greatest lower integer function.

The number of communication links required in the structure scales differently for each of the topologies. In the custom planar topology, the number of links required in the rightmost and leftmost bays is slightly less than the number of links for bays in the middle of the structure. When the size of the truss changes, there are still only one rightmost bay and one leftmost bay. Thus, changing the size of the truss changes the number of bays in the middle of the structure. Each bay in the middle of the structure requires twenty communication links for the custom planar topology, and each bay contains five elements. Thus, an average of four additional communication links are required for every element added to the truss.

In the custom hierarchical topology implementation of the ten-bay truss, the upper layer processors are completely connected. As the truss size increases, additional processors must be added to the upper layer of the hierarchy and it is no longer possible to completely connect all of the upper layer processors. Therefore, more layers must be added to the hierarchy in a tree structure to limit the number of links required in the upper layers. The way in which the number of communication links scales as the truss size increases depends upon the exact topology used to connect processing nodes in the upper layers. If a tree structure is used, however, the combined number of communication links in all of the upper layers will always be smaller than the number of communication links in the bottom layer. Thus, the number of communication links in the lower layer, which is known to scale linearly with the size of the truss, will be the dominant term in calculating the number of links in the custom hierarchical topology. For this reason, the number of communication links required in the custom hierarchical topology scales linearly with the size of the structure.

An $n$ processor hypercube implementation requires $n*\log_2 n$ communication links [83]. Since the number of processors scales linearly with the size of the truss, the number of communication links required scales as $N*\log_2 N$ with truss size, where $N$ is the number of elements in the truss.

The final cost factor that must be considered is the amount of embedded fiber needed to implement each of the topologies. The modularity of the custom planar makes it easy to see that each bay added to the truss requires an additional 22.5 units of embedded fiber. Thus, the cost of embedded fiber in the custom planar scales linearly with the size of the truss.

It is difficult to determine how the length of communication links in the custom hierarchical topology increases as the size of the network increases because several different hierarchical schemes could be employed as the size of the structure increases. As an example, Figure 5.8 shows a binary tree architecture used to implement the upper layers of the hierarchy for a truss with $J*K$ bays. The lowest two layers of the hierarchy consist of a group of $J$ portions of the truss. Each portion contains $K$ bays.

The number of communication links in the lowest layer of the hierarchy is much greater than the number of communication links in the upper layer. The lengths of the upper layer links, however, are greater than the lengths of the lower layer links. As the size of the truss doubles, the amount of embedded fiber needed to connect all but the top layer of the hierarchy also doubles. Figure 5.8 shows how the length of fiber required for each additional layer in the hierarchy increases as the number of bays in the truss increases. From this figure, it can be seen that if $L_{JK}$ units of embedded fiber are required for a $J*K$ bay truss, then $(2L_{JK} + 4K)$ units of fiber will be required for a $2*J*K$ bay truss.

To determine how the length of embedded fiber in the hypercube scales, the simplified truss model in which the bays contain only four truss elements is considered. Examining a one-bay truss indicates that 9.65 units of embedded fiber are required. A two-bay truss can be

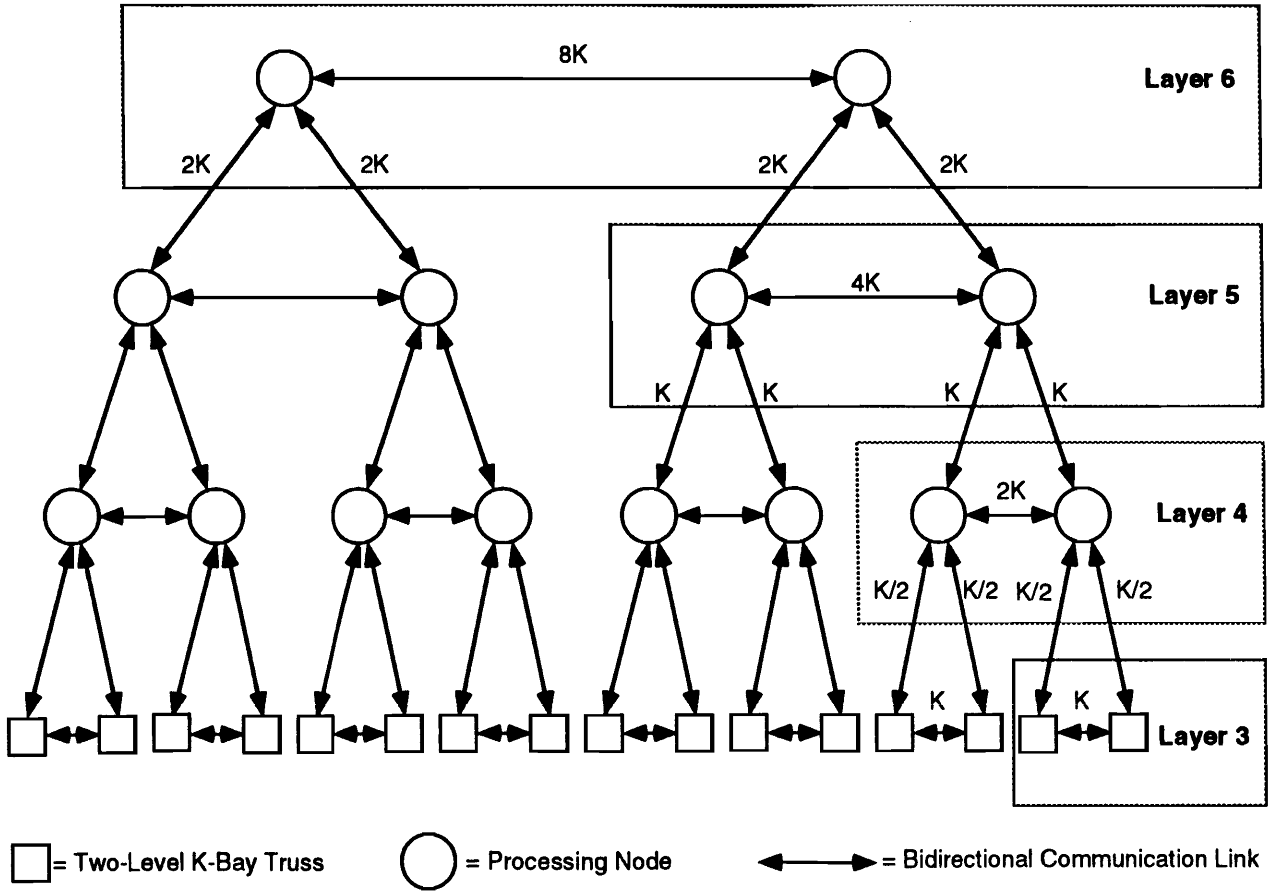


Figure 5.8. Binary tree connections for the custom hierarchical topology

formed by flipping an image of the one-bay truss around its vertical axis and joining it to the original one-bay truss. This is shown in Figure 5.9. An upper address bit is given to each element with the upper address bit being "0" for truss elements in the original one-bay truss and "1" in elements of the image. The amount of embedded fiber needed for the lower order address bits remains the same in each bay since these links remain internal to the bay. The amount of embedded fiber for the highest dimension communication link is 1 unit for each of the four horizontal connections and 1.414 units for each of the four diagonal connections.

Larger truss sizes can be constructed by continuing this process of joining an original $K$ bay truss with its vertical image and adding an extra address bit to form a $2K$ bay truss. The amount of embedded fiber is then calculated by doubling the amount of fiber required for the $K$ bay truss and then adding the amount of fiber required for the connections in the highest dimension links that join the two halves of the structure. For a $K$-bay truss with four elements per bay, there are $4K$ communication links in each dimension. Thus, there are $4K$ links in the highest dimension. $2K$ of the highest dimension links have an average length of $\frac{K}{2}$ while the other $2K$ links have an average length of ( $\frac{K}{2}$ + 0.414 ). Thus, given a $K$ bay truss with $L_K$ units of embedded fiber, it is known that a $2K$ bay truss will require $2L_K + 2K^2 + 0.818K$ units of embedded fiber. This shows that the amount of embedded fiber required increases quadratically with the size of the structure.

The five-element bay considered in this research will require more fiber than the four element bay discussed in the preceding paragraphs. The fact that the amount of additional embedded fiber increases quadratically with the size of the structure for a truss with four elements per bay means that the amount of embedded fiber will increase at least quadratically when trusses with five elements per bay are considered. Thus, a full hypercube implementation will become impractical as larger trusses are considered.

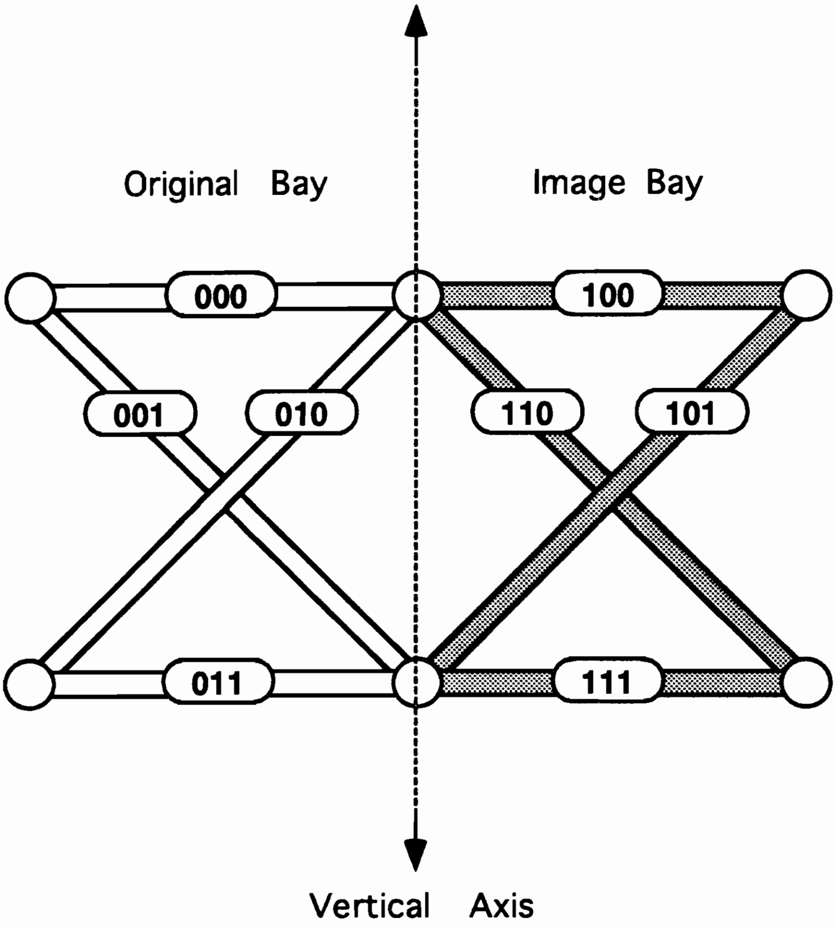


Figure 5.9. Construction of two-bay truss

### 5.5. Summary

This chapter described a damage DLE algorithm used for health monitoring in a ten-bay truss. The implementation of this algorithm on several different multicomputer architectures was presented. The costs associated with each of these implementations were examined. A discussion of the simulated performance of these architectures for a variety of operating conditions follows in Chapter 6.

# Chapter 6. Damage DLE Algorithm Simulation

This chapter discusses the simulation of a damage DLE algorithm on three different multicomputer topologies. The simulation was performed using the VHSIC Hardware Description Language (VHDL) [118,119]. The chapter begins with a discussion of the VHDL model. The simulation requires several timing parameters whose assigned values were based on other research. The reasoning behind each of these assignments is discussed. Next, the performance parameters that are considered in the simulation experiments are presented. The chapter concludes with a discussion of results.

## *6.1. The VHDL Simulation Model*

This section describes different facets of the VHDL model. It begins by defining how messages are represented and the routing mechanism used to transport these messages. Next, the communication link, processing element (PE), and communication controller (CC) models are discussed. The section concludes by describing data files created by a simulation run.

### 6.1.1. Model Definitions

Before the operation of the model can be understood, it is important to discuss the underlying mechanisms used in the communication network. The following discusses the header fields that are included with each message and the implementation of a wormhole routing mechanism that guides messages through the network.

#### 6.1.1.1. Message Fields

Header information must be included that identifies the important characteristics of each message. Figure 6.1 shows the header information that is used in this research. The first piece of header information is a status bit. This bit is set to signify that the message on the link is currently being transmitted and is cleared when the message transmission is completed. The header also includes information that tells where the message originated, the time the message was generated, which node the message is going to, and the length of the message. This message length is given in terms of how long it will take to transmit the message on a link. In an actual algorithm implementation, the message length would be given in terms of the number of bits the message contained.

Another piece of information a message carries is its algorithm flag. This flag signifies which step of the algorithm generated the message transmission. As was discussed in Chapter 5, the communication patterns for the damage DLE algorithm depend on the topology being used. For each topology, numbers are assigned to each step in the algorithm. When a processing element generates a messages that fulfills a given step in the algorithm, the algorithm step is included in the header information. This enables the node that receives the message to know why the message was transmitted and what information it represents. To go along with the algorithm flag, an algorithm period value is also included. This value tells the receiving node which iteration of the algorithm generated the message. If congestion exists and a message takes

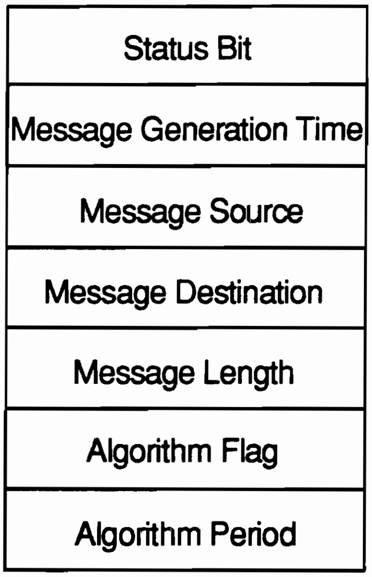


Figure 6.1. Message header fields

a long time to reach its destination, it may arrive after a given node has started its next processing cycle. The receiving node can look at the algorithm period flag and determine if the message is part of the current iteration or whether it is old information that should be discarded.

6.1.1.2. Wormhole Routing

In wormhole routing, all messages are broken up into individual packets that follow each other over the communication links. The lead packet contains information that states the source and destination of the message. When a processing node receives the first packet of a message, it examines the destination of the message and determines whether the message should transferred to the local processing element or to an outgoing link. If the link that the receiving node wants to transmit the message on is busy, the receiving node informs the sending node that the message is currently blocked and waits for the busy link to become idle. When the outgoing link becomes idle, the receiving node informs the sending node that it is routing the message to the next node in the message path. The transmitting node sends the next packet of the message to the receiving node as the receiving node transmits the first packet of the message to the next node in the message path. This process continues until the message reaches its destination. At this point, the entire message path is complete and the destination node informs all nodes in the message path that the rest of the message can be transmitted. The receiving node waits for the message transmission to complete, after which time it sets all of the links in the message path back to the idle state. Thus, each communication link in the system is in one of the four following states: idle, blocked, routing, or transmitting. A link is in the idle state when no nodes are attempting to transmit on it. A link in the blocked state is dedicated to a particular message, but the message transmission is halted because the message path is not complete. A link in the routing state is transmitting information that is attempting to establish the message path. Finally, a link in the transmitting state is sending data on a fully established message path.

### 6.1.2. Model Components

The computer networks discussed in this paper are simulated using structural models formed by the repeated instantiation of major system components. At the top level, the system is decomposed into the communication links and the processing nodes that use the links to share data. Each processing node, in turn, consists of a processing element (PE) and a communication controller (CC), as shown in Figure 6.2. The three interconnection network topologies examined in this research are constructed by altering the location of the communication links in the network, changing the routing mechanism that determines where messages travel in the network, and altering code in the processing elements that determines the steps of the algorithm that are being executed.

#### 6.1.2.1. The Communication Link Model

The communication links in the system are modeled as components that define which pairs of nodes are allowed to directly communicate with each other. To implement wormhole routing, a status signal from node $K$ to node $J$ must exist for each link from node $J$ to node $K$. This signal informs node $J$ of the condition of the message that it is sending to node $K$. A distinct pair of link definition variables exist for each link in the system. The first variable is a link status variable that takes on the values IDLE, ROUTING, BLOCKED, or TRANSMITTING depending on the state of the link. Details concerning the link status variable values are discussed further when the CC is described. The second link variable contains the header information that was described in Section 6.1.1.1. The CC sends messages over communication

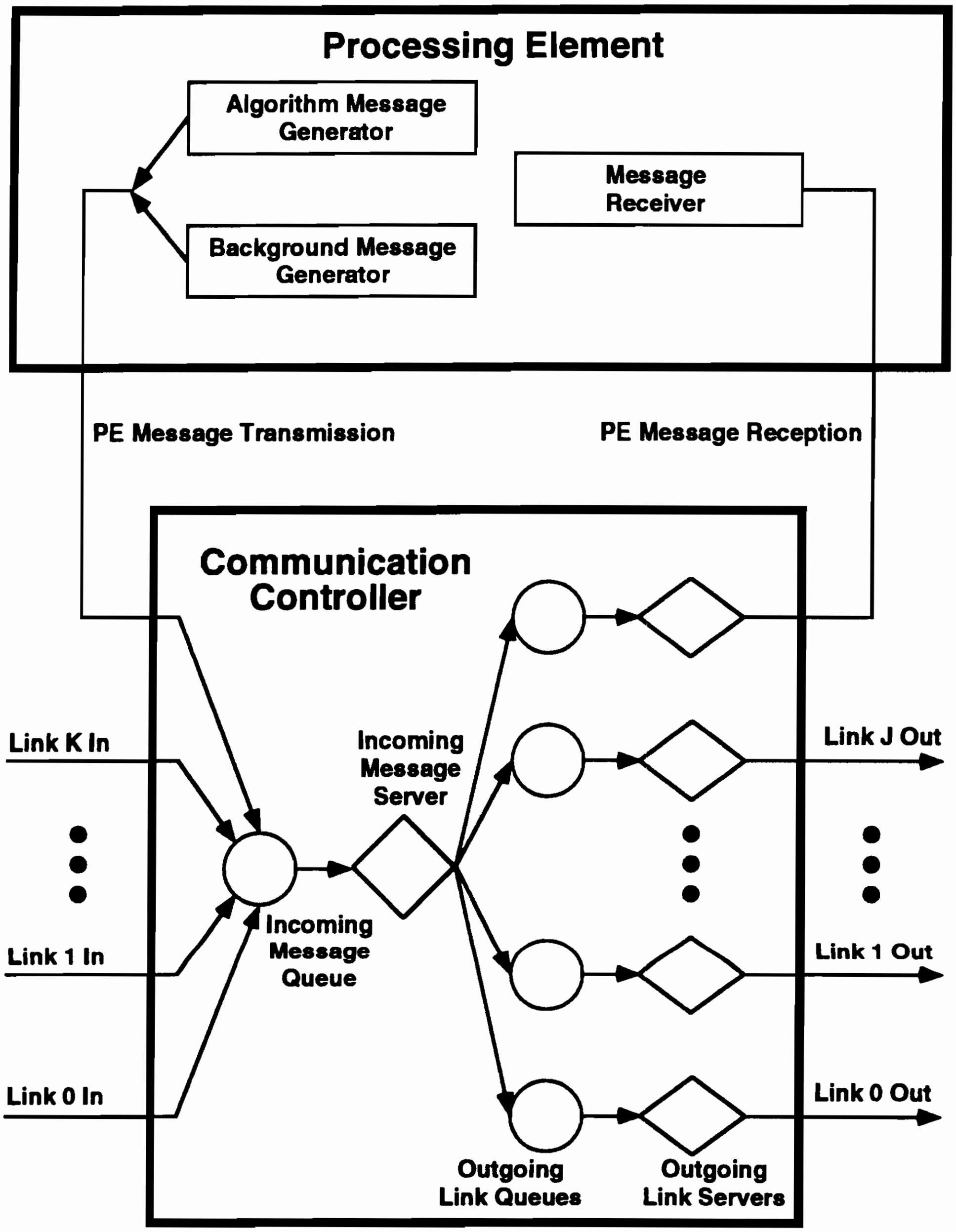


Figure 6.2. Processing node model

links by defining the values of the message field and link status variables. The location of links in the model is determined when the PE and CC models are instantiated in the VHDL program.

### 6.1.2.2. The Processing Element Model

All messages in the network are created by the PEs and sent to other PEs. A PE creates a message by defining the message length, which determines how long it takes to transmit the message, and the message destination, which determines which node will receive the message. Also, the PE tags the message with the time that it was created. This time tag is used for evaluating performance. Once a message is created, the PE sends it to the CC after a fixed time delay specified by the system parameter PE_TRANS_DEL. This delay simulates the time that it takes to transfer message data between the PE and the CC. The CC routes the message through the network to its destination. A thorough description of the CC operation is given in the next section.

In addition to generating messages, the PEs also receive messages. When a message arrives at the CC of its destination node, the CC sends the message to the node PE after a time delay. This delay simulates the time that it takes to transfer data between the CC and the PE. When the PE receives the message, the message is removed from the network and no longer affects the network performance. At the time the message is received, though, data concerning its travel through the network is obtained by observing when the message was created and when it arrived at its destination.

Two mechanism are used to generate messages in the network. Some of the messages represent data traffic that is generated by the damage DLE algorithm. Other messages are random background messages that are added to the network to determine the affects of additional traffic on performance. The PE message generation can be most easily described by discussing how each step of the algorithm is modeled in the PE. The exact PE operation changes depending on which topology is being simulated, but some basic steps of the PE operation are the same in all cases.

One node in the network is designated as the master node. An outside source is assumed to send this node data required by the algorithm, such as the initial mass and stiffness matrices, and the mode information required in each iteration. When operation begins, the master node broadcasts the mass and stiffness matrices to every node in the system. The amount of time required to transmit these matrices is given by the MASS_MAT_TRANS_TIME and STIFF_MAT_TRANS_TIME parameters, respectively. The algorithm flag in the message field of these broadcasts inform the receiving nodes what the data represents.

The iterative portion of the algorithm begins after an amount of time specified by the START_DEL parameter. This delay allows the mass and stiffness matrix information to propagate through the network. Each cycle of the iterative process begins at regular intervals specified by the parameter SAMPLE_PERIOD. The master node increments the algorithm period at the beginning of each iteration. The master node broadcasts the mode shape matrix and mode frequency vector for the new sample period to all processors in the system. The times required to transmit these matrices are given by the parameters MODE_SHAPE_TRANS_TIME and MODE_FREQ_TRANS_TIME, respectively. The master node sets the algorithm flag value of each message to tell the receiving processors what the data represents and the algorithm period value so that the receiving processors know which iteration the data belongs to. After transmitting the mode information, the master processor behaves like any other processor in the system.

When a processing node receives a message, it checks the algorithm flag to determine what the data represents and the algorithm period value to see whether the message is current. If the message is from an old iteration, it is discarded. If the message is from the next iteration and the current iteration has not yet completed, the node terminates any processing tasks it is executing on the current cycle and begins processing the next cycle. This is because once information from the next iteration is received, the processing node knows that the master node has begun the next iteration. Thus, the node knows that it has not completed its calculations or received all of the required data for the current cycle within the cycle time. Rather than try to complete the current cycle and catch up during the next cycle, the processor catches up by discarding any preliminary results it may have computed and proceeding directly to the next cycle.

If a message is received with the proper algorithm period value, then a flag is set in the processing node that records that the data carried by the message is available in the processor. The VHDL model simulates processing in the nodes using a series of wait statements and time delays. Initially, the processor waits to receive the mode information. Once the mode information is received, the processor waits for a time delay given by the DMG_EST_PROC_TIME parameter to simulate the time required to compute the damage estimate for the element in the structure associated with the processor.

After the damage estimate is computed, the processor transmits the damage information to other nodes in the system. The communication pattern that the processors use to share their damage estimates depends on the topology being implemented. The communication pattern used for each topology is described in Section 5.3.2. The time it takes to transmit a single damage estimate is specified by the SINGLE_DMG_MESG_TIME parameter. When groups of damage estimates are transmitted, the transmission time is equal to the number of damage estimates in the group times the amount of time required to transmit a single estimate.

A separate flag exists in each node for each step of the damage estimate sharing process. Since the number and sequence of steps in the sharing process depends of the communication topology, the meaning of the step flags also depends on the topology. As groups of damage estimates are received in the node, the appropriate flags are set to indicate the arrival of the data. After each step of data sharing is completed, the processor examines the flags to determine if the information required for the next step of the data sharing process has been received. If so, the processor proceeds with the next transmission. If not, the processor waits for the required data to be received.

When all of the damage estimates for a given iteration have been received, the processor updates the stiffness matrix for the truss based on the latest data and prepares itself for the next iteration. The MODEL_UPDATE_TIME parameter specifies the amount of time that it takes the node to perform this end-of-cycle processing.

In addition to algorithmically generated message traffic, background messages are also created by the processing nodes. The length of each message is randomly distributed according to the exponential distribution with an average length equal to the value of the system parameter MESG_LEN. The time between successive messages is also distributed according to the exponential distribution. The average time between generations is given by the parameter GEN_TIME. The average time between successive message generations is varied from one simulation to the next to examine how different loading conditions affect performance. The destinations of background messages are randomly chosen from a uniform distribution of all of the nodes in the network. Broadcasting is never used for background messages. The algorithm flag in the header field of a background message is set to 0. This allows receiving nodes to distinguish between background messages and algorithm-based messages. When a PE generates a background message, it sends the message to its CC. The CC routes the message to its destination in the same manner as algorithm-based messages.

#### 6.1.2.3. The Communication Controller Model

The CC is made up of an incoming message server, an outgoing message server for each of the outgoing links, and buffers that connect the links of the outgoing server to the network layer links. The number of incoming and outgoing links varies depending on the topology and processing node being represented.

The CC operation is the most complicated aspect of the simulation model. The node model, shown in Figure 6.2, shows the different pieces of the CC and the PE. As shown in the figure, the CC has incoming communication links, outgoing links, and connections to and from its corresponding PE. All of the incoming links are sent to a single incoming message queue. When a message arrives, it is placed in the message queue, and the status signal for the link that sent the incoming message is automatically set to BLOCKED. This informs the sending link that its message has arrived at the node but that it has not yet been serviced. If the message arrived from the local PE, no link status message is returned. A tag on the incoming message is set which states which link the message was received on or whether it was generated by the local PE.

The incoming message server constantly monitors the incoming link queue to determine if any new messages have arrived. The incoming message server examines the value of the destination field of the message of all incoming messages and determines whether the message has reached its destination. If the message is at its destination, it is placed in the received message queue. If the message is not at its destination, the server executes an algorithm that determines which outgoing link should be used to transmit the message to its destination. The next link algorithm is different for each of the topologies under consideration. The next link algorithm for each topology implements the routing rules discussed in Section 5.3.1.

The outgoing link servers monitor their corresponding link queues to determine if there are any messages to transmit. When the outgoing link server finds a new message, it undergoes a delay, whose length is specified by the parameter OUT_SERVER_DEL, to simulate the effects of the link control hardware preparing the message for transmission. It then sends the first packet of information, called a flit, over the link to the next node in the message path. A time delay specified by the parameter FLIT_DEL is inserted into this signal transfer to represent the transmission time of one flit. Also, the link server changes the status of the incoming link which it received the message from IDLE to ROUTING. This tells the previous links in the message path that the front end of the message has been sent to the next node in the message path. The link server then waits for the node that it sent the message to to respond by changing the status of the outgoing link that the message is transmitted on. From this point on, the outgoing message server takes any signal that it receives on the outgoing link status line and transmits it to the status line of the incoming link that the message was received on. This informs all preceding links in the message path of how far the message has traveled in the network. The status of the outgoing link will at first be set to a value of BLOCKED, and as the proceeding nodes in the network change the link status between BLOCKED, ROUTING, and eventually to XMITING, the outgoing link server continues to update the status of the incoming link which originally transmitted the message.

While the outgoing link server is updating its link status, it also keeps track of the time spent transmitting the message. When the outgoing message server receives the first bits of the message, it reads the length tag which indicates the time needed to transmit the message. Whenever the link status is ROUTING or XMITING, the outgoing message server is transmitting useful data on the link. Thus, as soon as the combined ROUTING and XMITING time is equal to the message length, the node knows that it has transmitted the entire message. At this time, the outgoing link status will be set back to IDLE and the outgoing message will once again examine its corresponding queue and wait for a new message to transmit.

The received message server acts like the outgoing link server, except that the message is known to have reached its destination. Thus, when this server receives the first packet of a message, it examines the length flag of the message to see how long it will take to receive the rest of the message. It sets the status of the receive link directly to XMITING for this amount of time to simulate the total transmission time of the message. Once the transmission is complete, the status of the incoming link returns to IDLE and the receive message server passes the message to the local PE. A time delay specified by the parameter PE_REC_DEL is inserted into the received message server at this point to represent the time it takes to transfer data between the CC and the PE. Once this has been completed, the received message server examines its queue and repeats this cycle for any new messages that have arrived.

### 6.1.3. Performance Monitoring

Many facets of the system operation can be used to judge the performance of the system. Two points of interest are explicitly examined by this research. First, the operation of the communication network is rated by examining the average delay of messages in the system and the standard deviation of this delay. Next, the time required to execute a single cycle of the DLE algorithm in each topology is compared. The data files produce by the simulation that indicate the network performance based on these parameters is discussed in this section.

Before information is written to any of the data files, the current time is compared to the START_STAT simulation parameter. The write operation takes place only if the current time is greater than the time specified by START_STAT. This helps to limit the size of the data files generated by the simulation runs. It also eliminates data generated before the system is in steady state.

#### 6.1.3.1. Message Delay Statistics

There are three types of messages that exist in the system: algorithm-generated broadcast messages, algorithm-generated direct-transmission messages, and randomly-generated direct-transmission background messages. The performance is affected by the message type. For this reason, the simulation outputs data that can be used to generate statistics for each type of message.

A file named "rxmesg.st" containing the necessary information for monitoring message passing performance is created during each simulation run. Each time a message is received at a processing node, a line is appended to the "rxmesg.st" file that contains the following information: the node that generated the message, the message destination, the time the message was generated, the time the message was received, the message length, the algorithm step that generated the message, and whether the message was a broadcast or a direct transmission. The algorithm step field of background messages is given a value of 0.

By examining the "rxmesg.st" file, it is possible to generate a number of statistics concerning the communication system. The average and standard deviation of message delay can be computed for each type of message in the system. Also, information can be gained regarding the average message length, the distribution of message generation within each cycle, and the number of messages required for each topology.

### 6.1.3.2. Cycle Time Statistics

The effectiveness of the topology for performing damage DLE is rated by the amount of time required to complete a single iteration of the algorithm. A given cycle is complete when all of the processing nodes have completed all computations for that cycle and have received all of the results generated by the other processing nodes.

Each simulation run generates a "cycle.st" file. When a processor completes an iteration of the algorithm, it appends a line to the end of this file that contains the following information: the current simulation time, the node number that is generating this data, the iteration number being completed, and an error flag. The value of the error flag depends on how the iteration completed. If the processing element completes all of its calculations and receives damage results from all of the other processors, then the error flag is set to 0. If the processing element does not complete the entire iteration when a message is received from another node for some future iteration cycle, then the processing element terminates all tasks it is performing for the current iteration, records cycle completion data to the "cycle.st" file with the error flag set to 1, and begins its processing tasks for the next iteration.

The data contained in the "cycle.st" file at the end of simulation indicates when each processor completed each iteration of the algorithm. The time at which the last processor completes a given iteration is the end of cycle time. The time at which the iteration began is subtracted from the end of cycle time to obtain the amount of time it takes to complete a single cycle. Since one simulation run contains several different cycles, the cycle period for all of the iterations is averaged to determine the mean cycle period for that simulation run.

## *6.2. Model Parameters*

The preceding paragraphs mention a number of model parameters that significantly affect the simulation results. The value assigned to each of the simulation parameters is shown in Table 6.1. This section discusses why each of the parameter values was chosen. The simulation parameters are divided into three categories. The first set of parameters values are a result of the hardware used in the communication system, the next set of parameters are a consequence of the message length and data passing requirements of the DLE algorithm, and the final set of parameters guide the simulation monitoring.

### 6.2.1. Communication Parameters

Four parameter values for the communication network are specified. The communication links in the system are assumed to have a capacity of 10 Mbps. This is consistent with the data rates that exist in many of today's supercomputers [69]. The FLIT_DEL parameter determines

Table 6.1. Simulation parameter values

| Communication Performance Parameters | |
|---|---|
| Parameter | Value |
| FLIT_DEL | 1000 ns |
| OUT_SERVER_DEL | 11 $\mu$s |
| PE_REC_DEL | 36 $\mu$s |
| PE_TRANS_DEL | 18 $\mu$s |

| Algorithm Parameters | |
|---|---|
| Parameter | Value |
| SINGLE_DMG_MESG_TIME | 12.88 $\mu$s |
| DMG_EST_PROC_TIME | 2.673 ms, 13.365 ms, or 66.825 ms |
| MODEL_UPDATE_TIME | 27 $\mu$s, 135 $\mu$s, or 675 $\mu$s |
| START_DEL | 200 ms |
| MASS_MAT_TRANS_TIME | 12.39 ms |
| STIFF_MAT_TRANS_TIME | 12.39 ms |
| MODE_FREQ_TRANS_TIME | 128 $\mu$s |
| MODE_SHAPE_TRANS_TIME | 281.6 $\mu$s |
| SAMPLE_PERIOD | 100 ms |
| MESG_LEN | exponentially distributed with a mean of 200 $\mu$s |
| GEN_TIME | exponentially distributed with mean varying between 0 and 1 messages per node per millisecond |

| Monitoring Parameters | |
|---|---|
| Parameter | Value |
| START_STAT | 400 ms |
| END_RUN_TIME | 700 ms |

how long it takes to send the first packet of information. A 32-bit message header is assumed, so it takes 3.2 $\mu$s to transmit this header on a 10 Mbps link.

The next set of communication parameters are associated with the interface hardware in the PE and CC of each node. Three delay parameters are used to model the operation of the communication hardware. PE_TRANS_DEL specifies the time it takes to send a message from a PE to its local CC, OUT_SERVER_DEL defines the amount of time it takes for a CC to determine where a received message should be routed to, and PE_REC_DEL specifies the time required to transmit a message from a CC to its local PE.

For systems that use wormhole routing, the time required to transmit a message can be broken up into the time required to create the message path and the time required to actually send the data. The time required to send the data is simply the length of the message divided by the data rate of the communication link. The time required to create the message path includes the time required to send data from the PE to the CC, the time required at each CC to determine where to route the message to, and the time required to send the message from the CC to the PE of the receiving node.

Berrendorf and Helin [69] examined the time required for message passing in Intel's iPSC/860 supercomputer. Although some of the timing information provided in their research depends on the average size of messages expected in the system, their results are used to choose realistic values for the message passing timing parameters of the VHDL model. The delay incurred by a message in an iPSC/860 for each hop in the message path for a message less than 100 bytes long is 11 $\mu$s. Thus, the OUT_SERVER_DEL parameter in the VHDL model is set to 11 $\mu$s. The startup time required to create a message path in the iPSC/860 for a message less than 100 bytes long was found to be 70 $\mu$s. This start up time includes the time it takes for the first communication controller to route the message. By subtracting the 11 $\mu$s required to initially route the message from the 70 $\mu$s startup time, the time required in the VHDL model to get the message from the transmitting PE to CC and from the receiving CC to PE, plus the time it takes to transmit the first flit of data, is found to be 59 $\mu$s. Thus,

$$\text{PE\_TRANS\_DEL} + \text{PE\_REC\_DEL} + \text{FLIT\_DEL} \approx 59\ \mu\text{s}.$$

The FLIT_DEL is 3.2 $\mu$s, so

$$\text{PE\_TRANS\_DEL} + \text{PE\_REC\_DEL} \approx 55.8\ \mu\text{s}.$$

Berrendorf and Helin did not compare the time required to send a message from a PE to a CC to the time required to send a message from a CC to a PE. The results of the testbed network discussed in Chapter 4, however, indicate that it took twice as long to receive communication data as it did to set up data for transmission. Thus, the VHDL model defines PE_TRANS_DEL as 18 $\mu$s and PE_REC_DEL as 36 $\mu$s.

### 6.2.2. Algorithm Parameters

The parameters discussed in this section deal specifically with the damage DLE algorithm. Many of the parameters define the time required to transmit messages containing the appropriate information for certain steps of the algorithm. To obtain a value for these parameters, the length of the message in bits is divided by the 10 Mbps transmission rate of the communication link.

The first values that the algorithm transmits are the mass and stiffness matrices for the truss. Both of these matrices have size $n_h$-by-$n_h$, where $n_h$ is the number of degrees of freedom in the truss. The ten-bay truss has 44 degrees of freedom, so both the mass and stiffness matrices contain 1,936 entries. Each entry in the matrix is assumed to be a 64-bit real value, so each of these matrix transmissions contains 123,904 bits. A 10 Mbps link can transmit this data in 12.39 ms. Thus, both MASS_MAT_TRANS_TIME and STIFF_MAT_TRANS_TIME are equal to 12.39 ms. These matrices are transmitted one time at the beginning of the simulation.

At each iteration, a mode shape matrix and a mode frequency vector are broadcast to all of the nodes. It is assumed that the algorithm detects damage by examining the first twenty mode shapes. Thus, the mode frequency vector has twenty real-valued entries, each of which is represented by 64 bits. This makes the MODE_FREQ_TRANS_TIME parameter equal to 128.0 $\mu$s. The mode shape matrix is a 44-by-20 matrix. Instead of broadcasting the entire matrix at one time, the rows of the matrix are broadcast separately. This is done because the mode shape matrix is a large matrix that takes a long time to transmit. The store-and-forward broadcast mechanism used in this research requires an entire message to be received before any part of the message is transmitted to the next node in the broadcast path. By breaking up the matrix into several smaller messages, the broadcasting transmission is pipelined and the information propagates through the network more quickly. Each row of the matrix contains forty-four 64-bit real values. A MODE_SHAPE_TRANS_TIME of 281.6 $\mu$s accounts for the time required to transmit each row of the matrix.

The remaining messages created by the algorithm are damage estimates generated by the truss elements. Each damage estimate contains two 64-bit values. The first value is the damage estimate itself. The second value contains status information, such as an element identifier and an indicator of whether this node has successfully performed all steps of the algorithm in previous iterations. The two 64-bit values require 12.8 $\mu$s to transmit. The number of damage estimates that are grouped together as the message propagates through the network depends on the communication topology and the step of the algorithm being executed. The transmission time for a group of $K$ damage estimates is $K$ times the time required to transmit a single estimate.

The only remaining issue for messages in the system is the average length of the background messages. It is reasonable that algorithms executed in the network would attempt to use message lengths comparable to those of other processes being executed in the system. Preliminary simulation runs indicated that the mean length of the algorithm-based messages is approximately 200 $\mu$s. Thus, the MESG_LEN parameter was set to 200 $\mu$s.

In addition to the communication system, the algorithm is also affected by the time it takes the processors to perform their computations. Over time, the ratio of processing time to communication time will continue to vary. This research examines the performance of the algorithm for several different computation to communication ratios. Initially, the processing time was set to 0 ns and the simulations were executed with no background traffic. The time required to complete a single iteration under these constraints is simply the time required for all of the message data to be passed between processors since the data processing is assumed to be performed instantaneously. The iteration time was measured to be 10.76 ms in the hypercube, 10.20 ms in the custom hierarchical topology, and 19.63 ms in the custom planar topology. The iteration time with instantaneous processing averaged over the three topologies is approximately 13.5 ms.

To examine how the communication-to-processing time ratio affects system performance, three different processing times are used in the simulations. Initially, the processing time is set equal to the average communication time. This examines the performance of the network when there is an equal tradeoff between processing and communication workload in the system. Next, the processing time is set to five times the communication time to examine the system performance when processing time is the dominant factor. Finally, the processing time is set to one-fifth of the communication time to examine the performance of a system in which data sharing takes the majority of the cycle time. Thus, the three processing time values used in the simulations are 13.5 ms, 67.5 ms, and 2.7 ms.

The processing time is divided between two tasks. First, the damage estimate is calculated after the mode information is received, and then the stiffness matrix is updated once all of the damage estimates are received. The calculation for the damage estimate requires many matrix multiply operations and several addition and subtraction operations. Updating the stiffness matrix simply requires that any of the new damage estimates that are different than the old damage estimate be replaced. The multiply operations would be expected to dominate the overall processing time. Thus, the DMG_EST_PROC_TIME parameter receives a value equal to 99 percent of the total processing time, and the remaining 1 percent of processing time is assigned to the parameter MODEL_UPDATE_TIME.

To determine an appropriate value for the START_DEL parameter, a temporary value of 500 ms was assigned. After 200 ms of simulation time, the mass and stiffness matrices transmitted at the beginning of the algorithm propagated to all of the processors in the system. Thus, the START_DEL parameter was assigned a permanent value of 200 ms. This represents enough time for all of the setup information to propagate to the processors.

The final algorithm parameter that needs to be assigned is the length of the sample period. This time represents the time for the system to complete a single iteration of the algorithm. The time required to complete an iteration of the algorithm is expected to be greatest when the processing time is the greatest. Thus, a processing time of 67.5 ms was used and the time required to complete an iteration of the algorithm was examined. It takes the hypercube 77.00 ms, 77.02 ms for the custom hierarchical topology, and 86.99 ms for the custom planar topology. The SAMPLE_PERIOD parameter was set to 100 ms. As background traffic is added to the network, these cycle times increase. The algorithm successfully executes if all of the processors complete the algorithm within this 100 ms. As traffic is added, its affects on the cycle time are monitored until the algorithm eventually fails because it requires more time to execute than is available in the sample period.

### 6.2.3. Simulation Monitoring Parameters

The final two parameters that need to be specified are the time that the gathering of statistical information should begin, given by START_STAT, and the time that the simulation should end, given by END_RUN_TIME. This study considers the steady-state behavior for each of the topologies. The most important output of the simulations is data concerning how long it takes to complete a single iterative cycle. As the system approaches steady-state, the expected time required to complete a single iteration should vary from one cycle to the next. When the system is in steady-state, the expected cycle times achieve constant value. The background message traffic causes some variance in the cycle times, but the average values for each iteration should not follow an upward or downward trend.

This research looks at several generation rates for background messages. The rates used in the research, as defined by the GEN_TIME parameter, were determined from preliminary work that determined the maximum generation rate the network could handle before it was no longer capable of transmitting the required messages within the cycle time and also determining the smallest message generation rate that had a noticeable affect on the cycle time. For steady-state analysis, system operation for the largest and smallest background message generation rates was examined. If the network reaches steady-state within a certain time for both the largest and smallest message generation rates, then it is expected that the network would also be in steady-state by this time for any message generation rate between these two values.

As with all of the simulation runs that use background messages in this research, three different random number seeds were used. The cycle times achieved for each topology, generation rate, and random number seed are shown in Tables 6.2a, 6.2b, and 6.2c. The iterative cycles do not begin until 200 ms into the simulation run. Before this time, setup information is transmitted in the network. New cycles begin every 100 ms. For some of the simulation runs, the iterative cycles starting at 200 ms and 300 ms do not complete in the

**Table 6.2a.** Hypercube steady-state analysis
(times in milliseconds)

| | Generation Rate (100 msg.) / (node * sec) | | | Generation Rate (1000 msg.) / (node * sec) | | |
|---|---|---|---|---|---|---|
| Cycle Start Time | Seed 1 | Seed 2 | Seed 3 | Seed 1 | Seed 2 | Seed 3 |
| 200 ms | 23.72 | 23.73 | 23.94 | X | X | 32.25 |
| 300 ms | 23.74 | 23.93 | 23.65 | 34.03 | 31.74 | 36.22 |
| 400 ms | 23.53 | 23.93 | 23.65 | 31.90 | 31.76 | 32.91 |
| 500 ms | 24.42 | 24.23 | 23.92 | 31.40 | 31.97 | 30.57 |
| 600 ms | 23.77 | 24.41 | 23.71 | 32.61 | 33.43 | 30.10 |
| 700 ms | 24.16 | 23.77 | 24.60 | 30.55 | 33.27 | 30.53 |
| 800 ms | 23.65 | 24.91 | 24.17 | 30.17 | 31.87 | 32.14 |
| 900 ms | 23.71 | 23.56 | 25.25 | 31.69 | 30.17 | 31.29 |
| 1000 ms | 23.70 | 24.37 | 23.69 | 32.58 | 31.90 | 32.80 |
| 1100 ms | 24.00 | 24.10 | 23.73 | 33.20 | 33.87 | 31.60 |

X = did not complete cycle

Table 6.2b. Custom planar topology steady-state analysis (times in milliseconds)

| | Generation Rate (100 mesg) / (node * sec) | | | Generation Rate (1000 mesg) / (node * sec) | | |
|---|---|---|---|---|---|---|
| Cycle Start Time | Seed 1 | Seed 2 | Seed 3 | Seed 1 | Seed 2 | Seed 3 |
| 200 ms | X | X | X | X | X | X |
| 300 ms | 34.73 | 35.10 | 35.35 | X | X | 41.11 |
| 400 ms | 35.46 | 35.07 | 36.22 | 39.46 | 35.93 | 37.83 |
| 500 ms | 34.25 | 35.00 | 35.46 | 36.56 | 39.84 | 36.50 |
| 600 ms | 35.94 | 38.61 | 35.91 | 43.10 | 40.48 | 38.80 |
| 700 ms | 35.14 | 36.51 | 34.71 | 37.81 | 40.11 | 37.50 |
| 800 ms | 36.15 | 35.20 | 38.45 | 42.61 | 38.86 | 39.11 |
| 900 ms | 35.89 | 34.92 | 36.34 | 37.80 | 36.87 | 38.41 |
| 1000 ms | 35.61 | 35.88 | 36.37 | 39.17 | 39.85 | 38.02 |
| 1100 ms | 34.56 | 36.00 | 36.69 | 37.58 | 39.86 | 38.04 |

X = did not complete cycle

Table 6.2c. Custom hierarchical topology steady-state analysis
(times in milliseconds)

| | Generation Rate (100 mesg) / (node * sec) | | | Generation Rate (1000 mesg) / (node * sec) | | |
|---|---|---|---|---|---|---|
| Cycle Start Time | Seed 1 | Seed 2 | Seed 3 | Seed 1 | Seed 2 | Seed 3 |
| 200 ms | 24.92 | 24.30 | 24.61 | 27.97 | 26.82 | X |
| 300 ms | 24.53 | 24.38 | 24.18 | 26.31 | 27.71 | 26.47 |
| 400 ms | 24.77 | 24.96 | 24.33 | 26.13 | 26.48 | 27.41 |
| 500 ms | 24.25 | 24.20 | 24.32 | 26.97 | 26.08 | 25.96 |
| 600 ms | 24.51 | 24.27 | 24.13 | 27.41 | 26.62 | 26.70 |
| 700 ms | 24.08 | 24.53 | 24.18 | 27.75 | 28.31 | 27.07 |
| 800 ms | 24.64 | 24.05 | 24.04 | 26.92 | 25.49 | 26.59 |
| 900 ms | 24.53 | 24.03 | 26.07 | 31.00 | 28.68 | 28.45 |
| 1000 ms | 23.85 | 24.73 | 24.79 | 27.30 | 27.52 | 27.36 |
| 1100 ms | 24.34 | 24.09 | 25.27 | 26.36 | 25.71 | 26.23 |

X = did not complete cycle

allotted sample period. This is because the transmission of the large mass and stiffness matrices at startup cause message congestion. This congestion backs up the system queues and it takes some time before they are emptied.

Output data concerning the network operation should not be collected until the system is in steady state. All of the iterative cycles that begin later than 400 ms into the simulation successfully complete their iteration cycles. Also, there is no trend in the cycle times recorded after 400 ms that indicates that the simulation is not in steady state. Since all three of the topologies appear to be in steady state after 400 ms, the START_STAT parameter is set to 400 ms.

The simulations contain random traffic, so the performance of the network for one iteration is not expected to be exactly the same as the performance in the next iteration. The true performance of the network can be more effectively examined by averaging the performance of several iteration cycles. This research averages the cycle time for three cycles to determine the mean cycle time for a given simulation run. It is not appropriate to examine the simulation over a portion of an iteration cycle because the network loading during one part of the cycle may be significantly different than the network loading at some other part of the cycle. Thus, the time between the START_STAT and END_RUN_TIME parameters should contain an integer number of iterative cycles. Since the START_STAT parameter is set to 400 ms, and the three cycles require 100 ms each to complete, the END_RUN_TIME parameter is set to 700 ms. By this time, the required data for three iterative cycles has been collected.

### *6.3. Simulation Results*

The results of the simulation are divided into two sections. The cycle time gives an indication of the overall system performance, while the message statistics provide insight into the finer details of the system operation. Initially, all of the topologies were examined with no background messages, as specified by setting GEN_TIME to zero. Next, preliminary results showed that generation rates below 0.05 messages per node per millisecond had little affect on system performance. Thus, a GEN_TIME equal to 0.05 messages per node per millisecond is examined to show the initial affects of background message traffic on system performance. Next, it was found that as the message rate rose slightly above 0.1 messages per node per millisecond, the custom planar topology was no longer able to complete its processing cycles. This is the third GEN_TIME value used in the experiments. The fourth and fifth GEN_TIME values were determined by examining what was the largest message rate that could be handled by custom hierarchical and hypercube topologies. These rates were found to be 0.2 and 1.0 messages per node per millisecond, respectively, for the custom hierarchical and hypercube topologies. For this reason, the data in the graphs presented in this section are plotted against GEN_TIME values of 0, 0.05, 0.1, 0.2, and 1.0 messages per node per millisecond. Results for the cycle time and message delay analysis are presented below. The data used in the plots discussed in the following sections, as well as statistical confidence interval for each of the data points, are presented in Appendix A.

### 6.3.1. Cycle Time

The overall effectiveness of each of the topologies for implementing the damage DLE algorithm can be judged by examining the time required to complete a single iteration of the cycle. This time includes processing time, communication delays, and delay caused by data dependencies in the network.

Figure 6.3a plots the cycle time versus the message generation rate for a processing time of 2.7 $\mu$s. The figure shows that the cycle times increase linearly as the message generation rate increases until the network is no longer able to support more background traffic. As was noted in the discussion of wormhole routing contained in Section 2.2.3.1, as long as the network is not

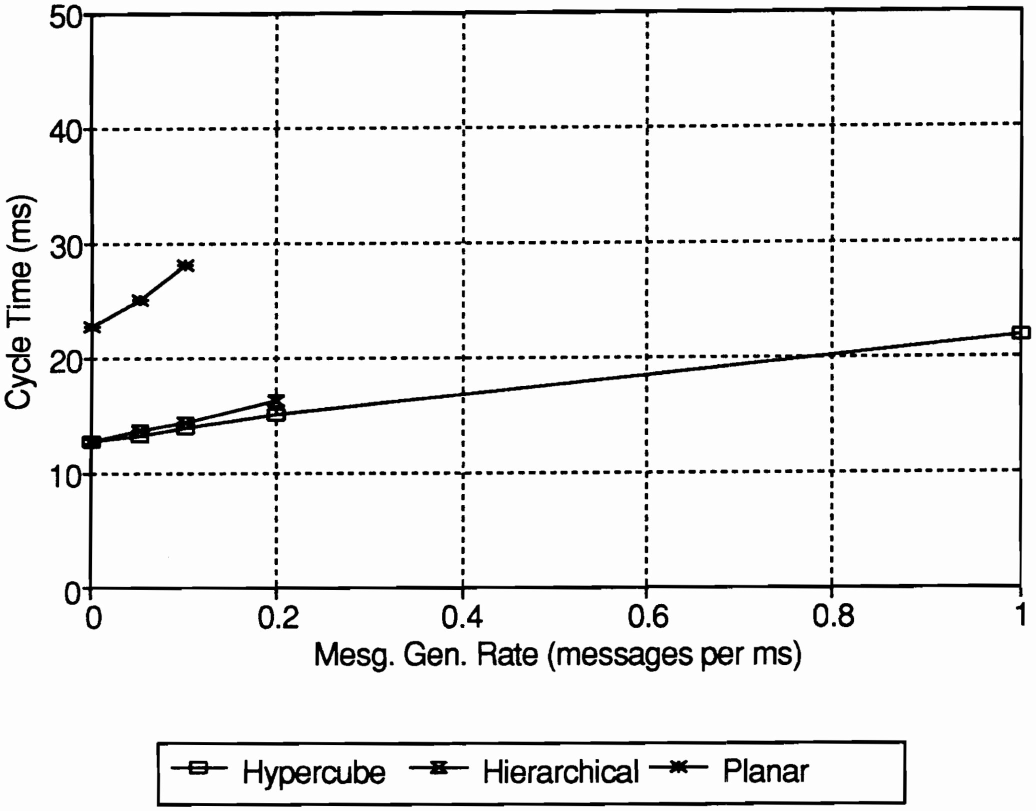


Figure 6.3a. Damage DLE algorithm cycle time for a processing time of 2.7 $\mu$s

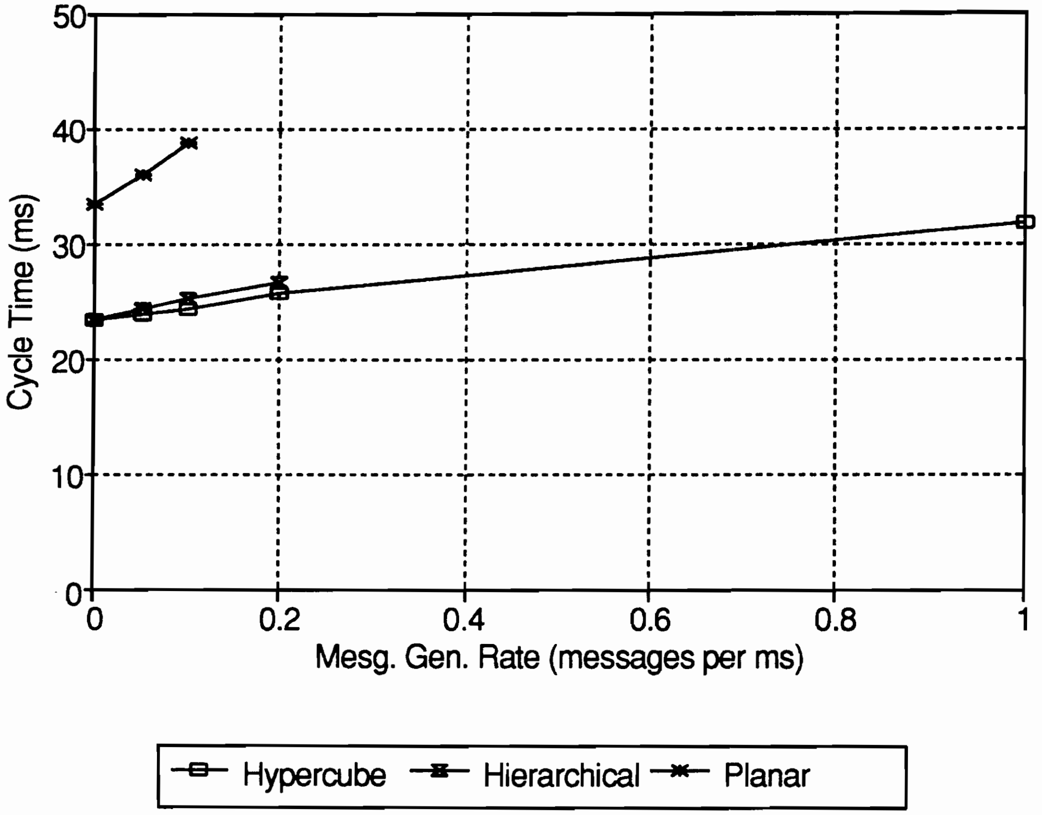


**Figure 6.3b.** Damage DLE algorithm cycle time for a processing time of 13.5 $\mu$s

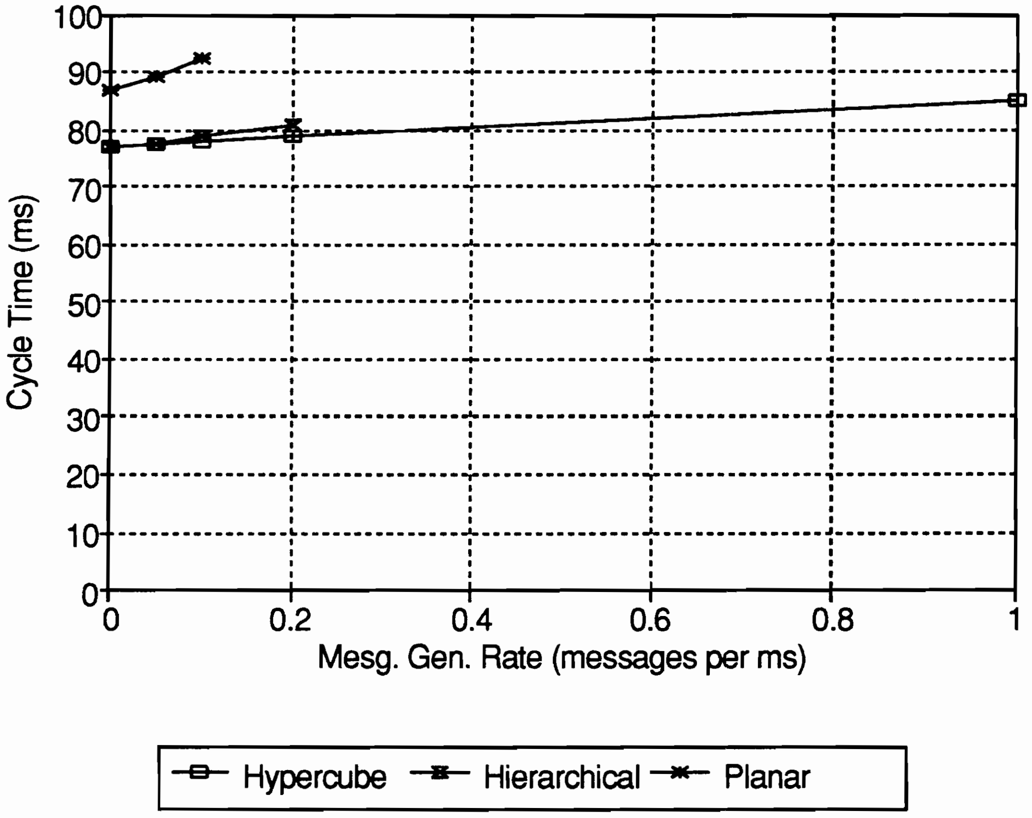


Figure 6.3c. Damage DLE algorithm cycle time for a processing time of 67.5 $\mu$s

saturated, increasing message generation rates cause only small increases in message latency. When the message generation rate is large enough to bring the network close to saturation, small increases in the message generation rate cause large increases in message latency. Thus, data that accurately map the increasing cycle time when the network is near saturation were not obtained.

The custom planar topology only supports a random generation rate of up to 0.1 message per node per millisecond. Also the cycle time for the custom planar topology at all of the message rates that it supports is about 10 ms longer, or approximately twice as long, as the cycle times for the other two topologies. The custom hierarchical and hypercube topologies have approximately the same cycle time for all message rates that they both support. The custom hierarchical topology, however, becomes saturated at a generation rate of 0.2 messages per node per millisecond while the hypercube does not saturate until a rate of 1.0 messages per node per millisecond.

Figures 6.3b and 6.3c plot the cycle time versus the message generation rate for processing times of 13.5 $\mu$s and 67.5 $\mu$s, respectively. The plots for the three topologies retain the linear characteristic shown in Figure 6.3a. Also, the relative values of the cycle times, with the custom hierarchical and hypercube topologies approximately equal and the custom planar topology approximately 10 ms longer than them, remains the same. As the processing time increases, however, the cycle time also increases. Thus, the 10 ms difference between the custom planar topology and the other topologies doubles cycle time for the custom planar topology when the processing rate is 2.7 $\mu$s, but the custom planar topology cycle time is only about 10 percent larger when the processing time is 67.5 $\mu$s.

A less obvious result displayed by these statistics is that increasing the cycle time by 10.8 $\mu$s, from 2.7 $\mu$s to 13.5 $\mu$s, does not raise the cycle time by 10.8 $\mu$s as might be expected. This shows that the processing and communication times alone do not determine the action of the system, but rather the interaction of processing and communication determine the overall cycle time. The interaction of the communication and processing systems can be more easily visualized by examining the message delay statistics.

### 6.3.2. Message Statistics

The cycle time results demonstrate the dependence of the cycle time on the processing time. The message passing delays, however, show little dependence on the processing time. Thus, this section presents these results only for simulations using a processing time of 13.5 $\mu$s.

#### 6.3.2.1. Global Message Statistics

The first message passing result examined in this research is the average message passing delay versus background message generation rate shown in Figure 6.4. Across the range of background generation rates, the hypercube exhibits the smallest average message delay and the custom planar topology exhibits the largest. The message delay in the hypercube is seen to be about half as long as the delay in the custom planar topology. The message delay in the custom hierarchical topology, however, is just slightly longer than the hypercube delay.

At first glance, it seems strange that the average message passing delay becomes smaller as the generation rate for background messages increases. The reason for this decline, however, is that there are different classes of messages in the network. The store-and-forward mechanism used to route broadcast messages causes these message to require more time to reach their destination. The mean delay of broadcast messages is shown in Figure 6.5. Thus, the higher the percentage of broadcast messages in the system, the higher the average message delay. As the generation rate for background messages increases, the average delay for each message type may increase, but the average delay on a per message basis decreases because more of the

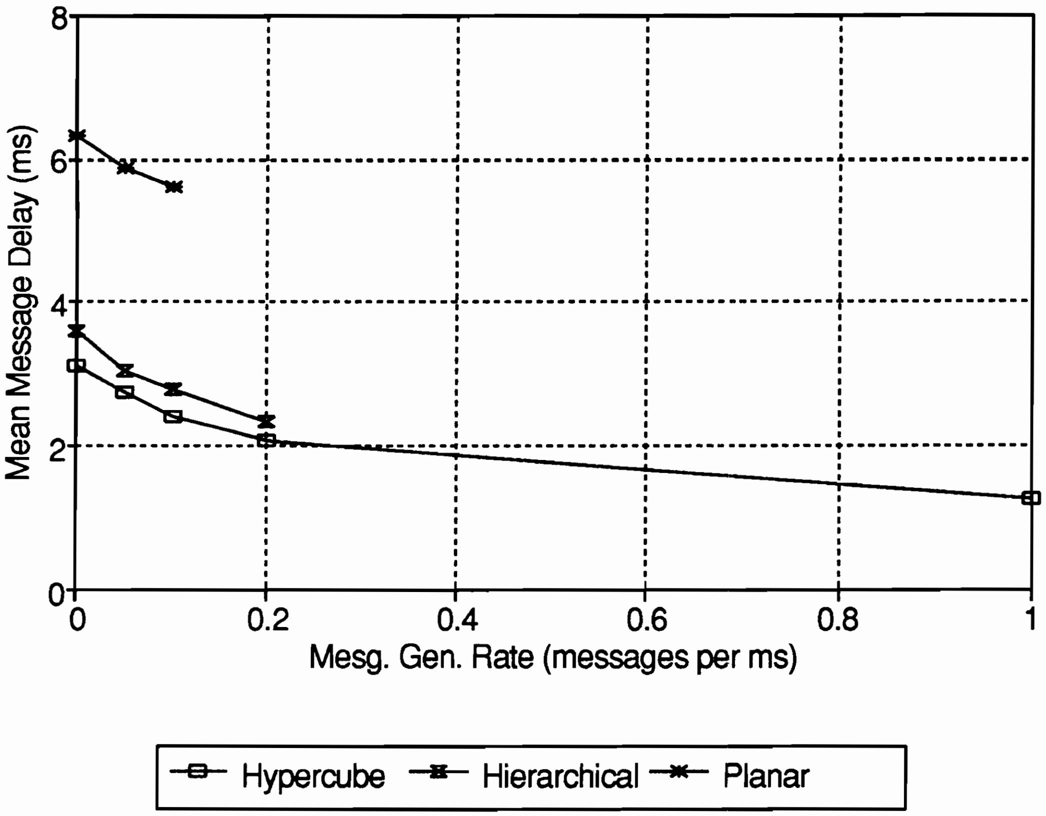


Figure 6.4. Mean message delay

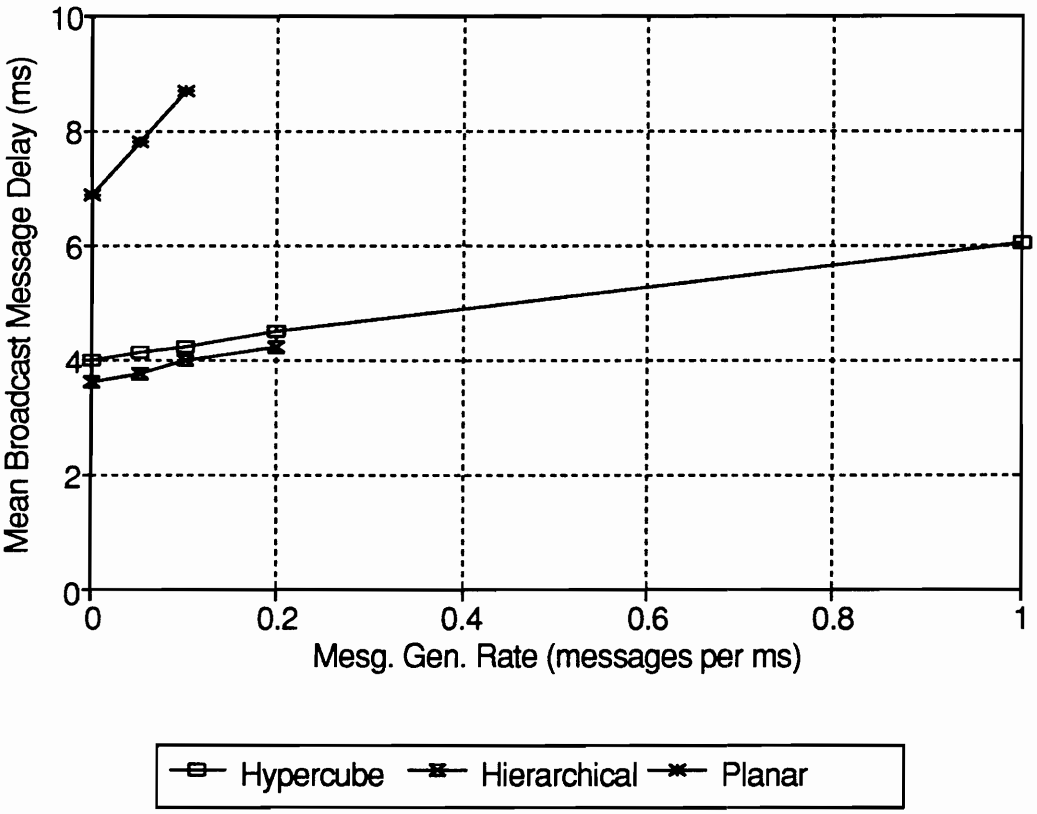


Figure 6.5. Mean delay for broadcast messages

randomly generated messages, which typically have the shortest message delay, are considered in the analysis.

### 6.3.2.2. Background Message Statistics

The last section discussed the expected behavior of the average delay for background messages. The data shown in Figure 6.6 supports these findings by showing that the average delay for background messages is approximately ten times smaller than the average delay for all messages in the system.

Figure 6.6 also shows that the amount of delay rises as the background message generation rate increases. It shows that for all background message generation rates, the hypercube exhibits the smallest message delay and the custom planar topology exhibits the largest. The custom hierarchical topology, which exhibits nearly the same system-wide message delay as the hypercube, does not perform as well when only the background messages are considered.

Figure 6.7 shows the standard deviation of message delay versus the background message generation rate. Some of this deviation can be attributed to the fact that the destination of the background messages is randomly chosen, and thus the amount of delay encountered in the message path is biased by the varying length of the message path. The degree of this bias is expected to be greatest in the topologies that have the largest network diameter, and thus the most difference between the shortest message paths and the longest message paths. From Figure 6.7, it can be seen that the hypercube, with a network diameter of 6, has the smallest standard deviation of message delay while the custom planar topology, with a network diameter of 12, has the largest. The custom hierarchical topology has a network diameter of 5, but the distribution of the distances between nodes is different than for that of the hypercube. Over 75 percent of the nodes in the hypercube are between two and four hops away from any given node. In the custom hierarchical topology, however, distances between nodes are more uniformly

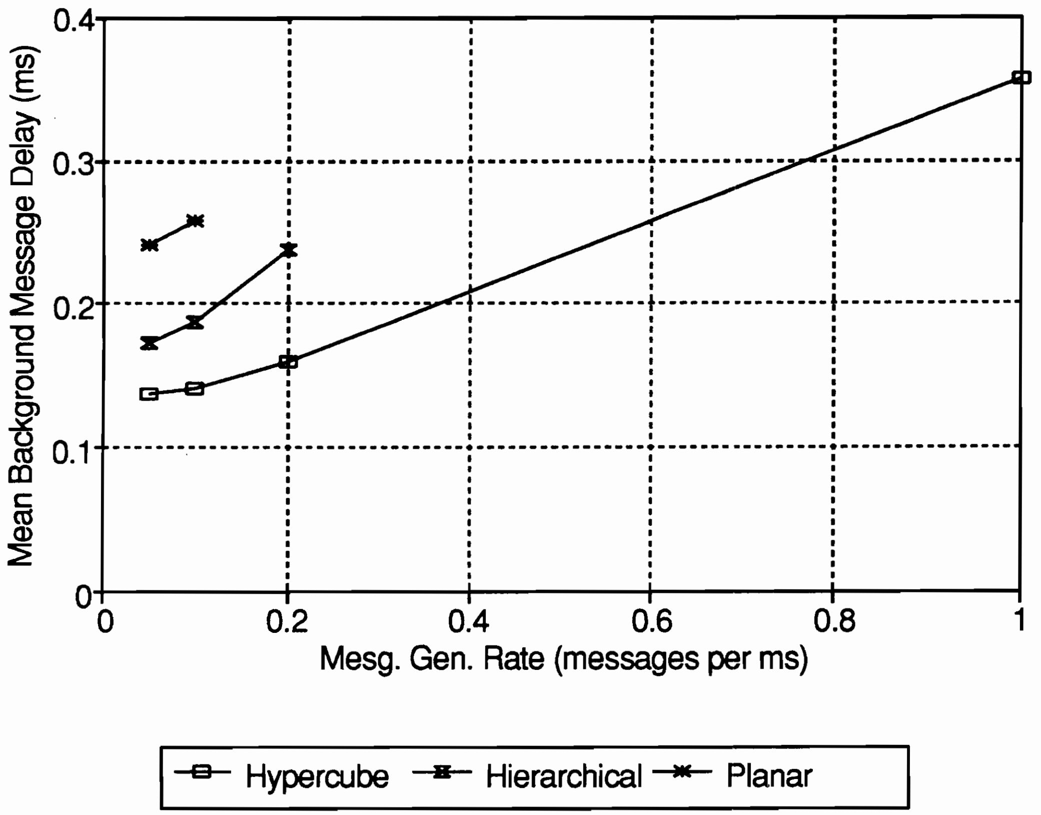


Figure 6.6. Mean delay for background messages

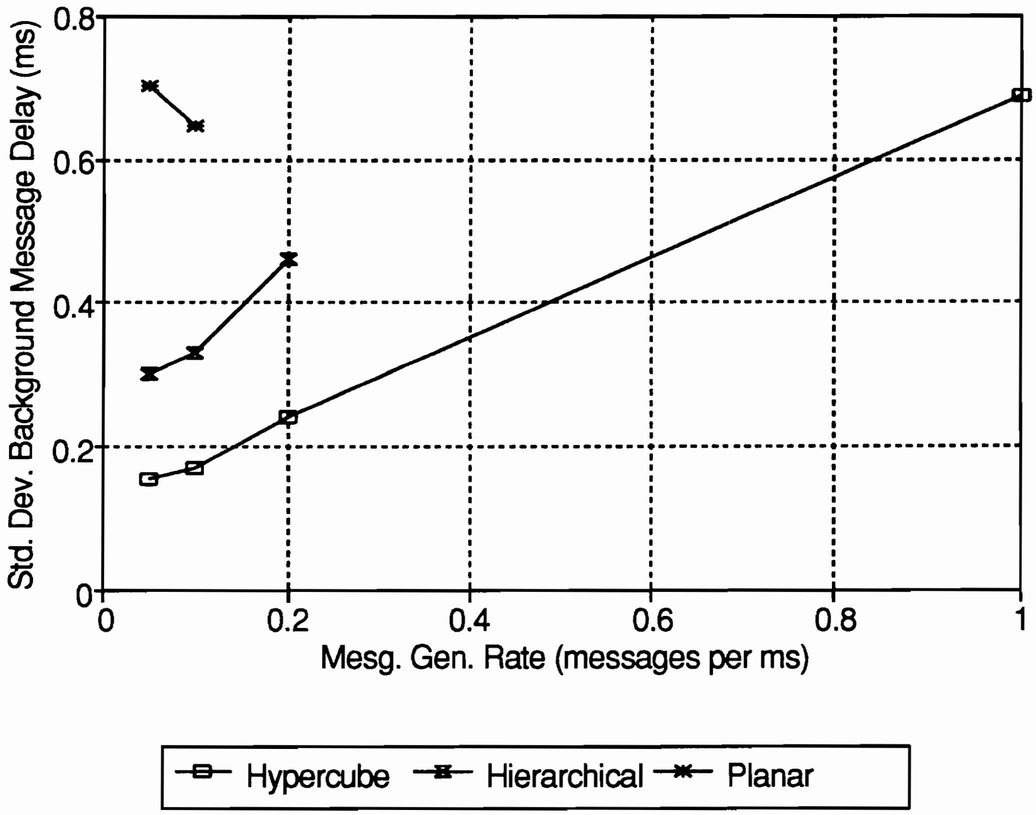


Figure 6.7. Standard deviation of delay for background messages

distributed between the highest and the lowest values. This distribution is one factor that causes the standard deviation of delay for background messages to be large even though its network diameter is smaller than that of the hypercube.

Another interesting feature shown in Figure 6.7 is that the standard deviation of message delay for the custom planar topology decreases as the background traffic generation rate increases. Typically, as the network becomes more congested the standard deviation of message delay would be expected to increase. By examining the data shown in Appendix A, however, it can be seen that the drop in the standard deviation of message delay in the custom planar topology for background messages is not statistically significant. The confidence interval around the two data points is large enough in this case to make it unclear as to which value is actually larger.

#### 6.3.2.3. Algorithm-Based Message Statistics

The previous figures have demonstrated that the delay for algorithm-based messages is much larger than the delay for the background messages. Upon further examination, however, it can be shown that the large delay for the algorithm-based messages is skewed by the existence of broadcast messages used in the algorithm. Also, it has been shown that the difference in message delays between the topologies is affected by the distance the message travels in the network.

To examine the performance of direct transmission messages in the system that travel the same distance in the network for all of the topologies, the message delay for non-broadcast algorithm-based messages is examined. For all of the topologies and algorithm implementations examined in this research, non-broadcast algorithm-based messages are always transmitted between neighboring nodes. The average message delay for these messages versus the generation rate of background traffic is plotted in Figure 6.8.

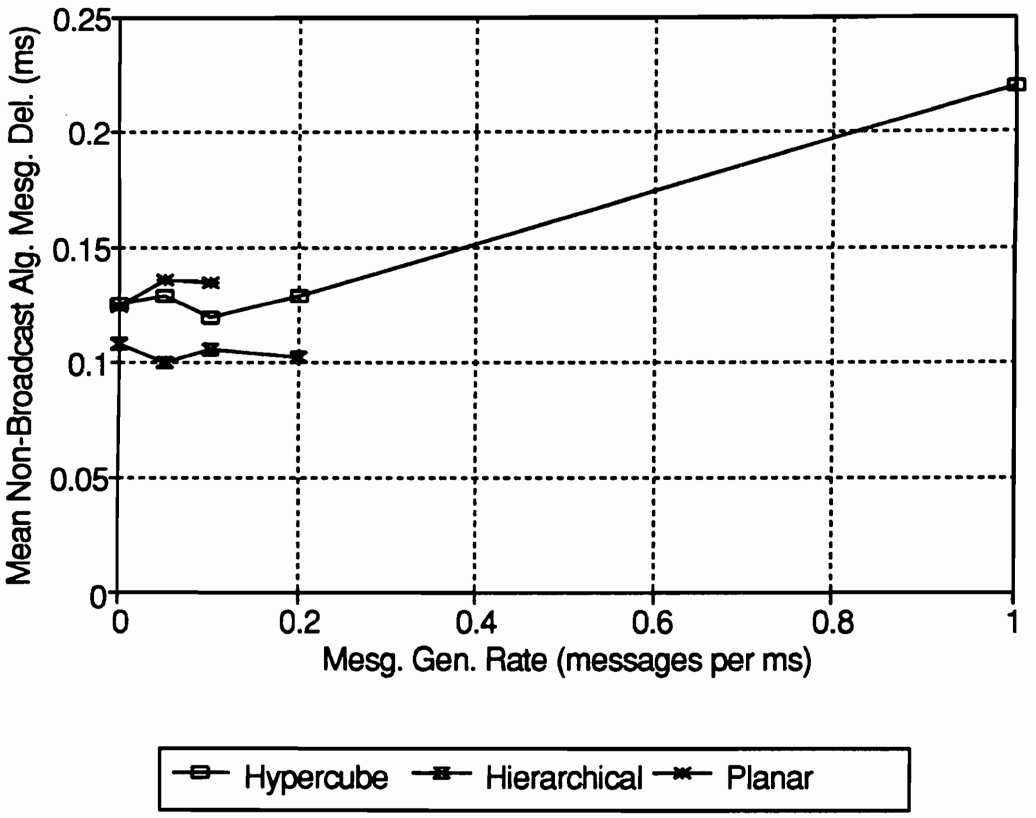


Figure 6.8. Mean delay for non-broadcast algorithm-based messages

For generation rates between 0 and 0.1 messages per node per millisecond, all three topologies have approximately the same average message delay. The difference between the average delay for the hypercube and the custom planar topology is statistically insignificant, and the custom hierarchical topology shows only a slightly smaller delay.

The most interesting result shown in Figure 6.8 is that the average delay does not rise by a statistically significant amount, and in fact may even fall, when the background message rate begins to increase. This fact can be explained by examining the relationship between the communication and processing times. If the communication delay is small, all of the processing nodes receive the initial setup data at about the same time, take the same amount of time to process the data, and attempt to share results at the same time. This causes congestion in the network at the instances where many different nodes are trying to transmit even though the network is relatively uncongested at other times in the sample period. Thus, the average message delay for algorithm-based messages is somewhat larger than the message delay than would be expected at other points in the cycle.

The addition of background message traffic causes some of the processors to wait for data before they can begin their processing cycle. Results from these processors are delayed and do not enter the communication network until the congestion from other algorithm-based message transfers has subsided. Thus, the start time of the message transmissions are delayed, but the average transmission delay incurred by the messages drops. As the background message generation rate continues to rise, the background messages become the dominant factor causing network congestion and the delay for all messages increases.

The affect that background messages have on spreading out the algorithm-based message generation times can be seen in Table 6.3. In this table, the 100 ms iteration period is divided into 1 ms intervals. The percentage of non-broadcast algorithm-based messages for the cycle that were generated during each of the 1 ms intervals is recorded. These traffic percentages are

Table 6.3. Distribution of message transmission times

Percentage of direct-transmission algorithm-based messages originating in each 1 ms block of the iteration period in the custom hierarchical topology for three background message generation rates.

| Start Time of 1ms Block | Background Message Generation Rate (messages per node per millisecond) | | |
|---|---|---|---|
| (in milliseconds) | 0.05 | 0.10 | 0.20 |
| 0 to 18 | 0.00 | 0.00 | 0.00 |
| 19 | 2.63 | 2.34 | 0.88 |
| 20 | 5.26 | 2.05 | 3.51 |
| 21 | 36.55 | 16.08 | 4.68 |
| 22 | 19.59 | 24.27 | 14.04 |
| 23 | 34.80 | 17.54 | 20.18 |
| 24 | 1.17 | 37.72 | 16.67 |
| 25 | 0.00 | 0.00 | 28.36 |
| 26 | 0.00 | 0.00 | 11.70 |
| 27 to 99 | 0.00 | 0.00 | 0.00 |

computed for the custom hierarchical topology with message generation rates of 0.05, 0.1, and 0.2 messages per node per millisecond. From this table, it can be seen that the times at which algorithm-based messages are generated spread out as the background message generation rates increase. For a background message generation rate of 0.05 messages per node per millisecond, all of the algorithm-based direct transmission messages are generated between 19 ms and 25 ms into the iteration cycle. Also, up to 36 percent of the algorithm-based message traffic is generated in the same 1 ms interval. When the background message generation rate is increased to 0.2 messages per node per millisecond, the algorithm-based direct transmission messages are generated between 19 ms and 27 ms into the iteration cycle, and no one 1 ms sample interval generates more than 29 percent of the algorithm-based message traffic.

It is also interesting to examine the message passing delay for the direct transmission algorithm-based messages versus the delay of background messages. For the hypercube topology, the mean delay for algorithm-based messages is about the same as the mean delay for the background messages. For the custom planar and custom hierarchical topologies, the mean delay for background messages is approximately twice as long as the mean delay for algorithm-based messages. This shows that the fact that the background messages may require several hops to get to their destination significantly affects performance in the custom topologies, but does not seem to be as pronounced in the hypercube. Thus, a notable drop in performance is observed as the distance a message must travel in the custom topologies increases, whereas the hypercube accommodates changes in the message path length more effectively.

The standard deviation of delay for direct transmission algorithm-based message is shown in Figure 6.9. Only a small increase in the standard deviation of message delay is observed as the background message generation rate increases. This is once again a consequence of the fact that the background messages spread out the transmission times of the algorithm-based

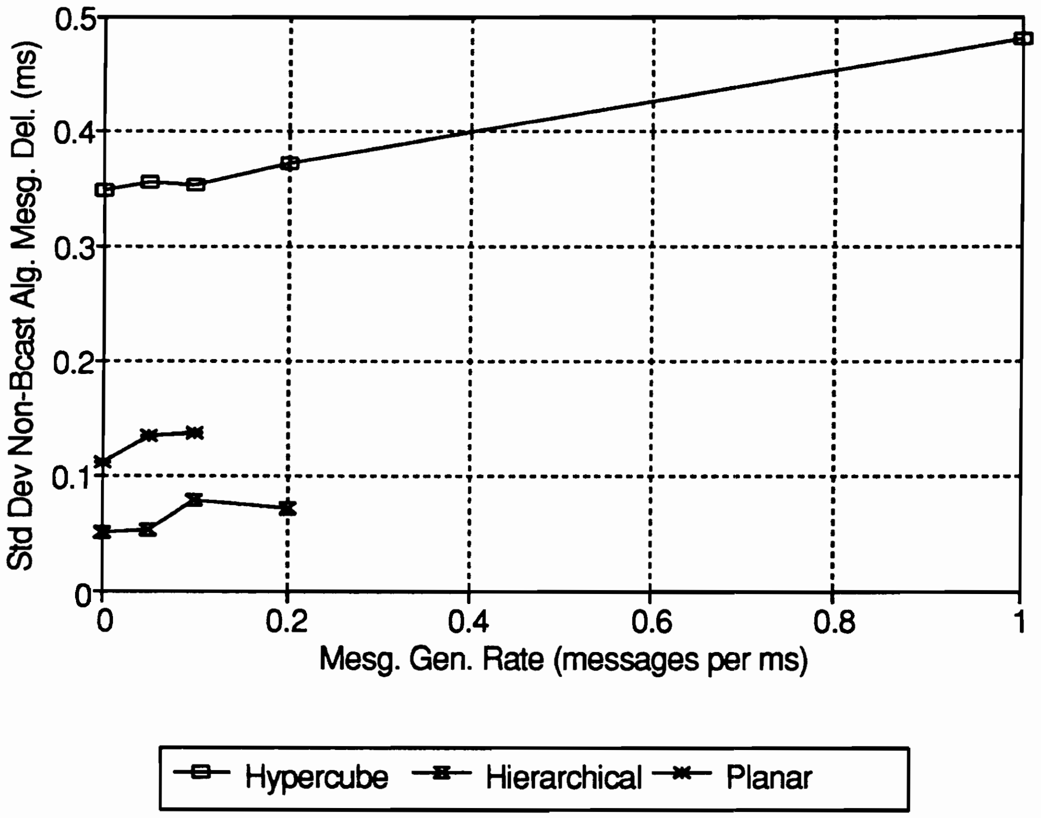


Figure 6.9. Standard deviation of delay for non-broadcast algorithm-based messages

messages, and thus lower the congestion in the network and the message delay for individual transmissions.

A surprising result shown in Figure 6.9 is that the standard deviation is approximately three times as large in the hypercube than in the other two topologies. This occurs despite the fact that all of the topologies had approximately the same mean message passing delay. Also, the standard deviation of delay for background message traffic in the hypercube was smaller than for the other two topologies. As was discussed earlier, one reason the hypercube has a smaller standard deviation for background message delays is that the average distance that a background message must travel in the network is somewhat consistent. For the algorithm-based messages, all of the messages travel over only one link, and thus the effects of the different path lengths are erased. This helps to diminish the effectiveness of the hypercube relative to the other topologies.

The two custom topologies contain many localized communication links. Thus, the affects of background traffic may remain somewhat isolated from the intrabay communication used in the algorithms. For this reason, algorithm-based messages are not heavily influenced by background traffic and operate with more predictable message delays. The links in the hypercube are based on a six-dimensional space with uniform communication links in all directions of the network. This causes the background messages travelling through the network to have a greater chance of blocking message transmissions between adjacent nodes in the hypercube. Thus, the standard deviation of message delay for the algorithm-based messages is greater in the hypercube because the message passing delay is influenced by the random effects of the background messages.

### *6.4. Summary*

This chapter discussed simulation of a damage DLE algorithm on three multicomputer topologies. It described the important characteristics of the VHDL model used in the simulations and examined several variable network parameters. The discussion of the simulation results, however, concentrated on how the multicomputer executed this particular damage DLE algorithm. The next chapter expands the conclusions drawn from the simulation results by discussing their global significance with regard to smart structures.

# Chapter 7. Conclusions

This research examined the implementation and use of multicomputers for distributed processing in smart structures and highlighted operational characteristics of these systems. The literature discussed in Chapter 2 describes reasons why distributed processing is well suited for smart structures and exposes desirable characteristics that a computer system for a smart structure should possess. Features of multicomputers in smart structures that differ from those of multicomputers in other applications were also presented. Chapter 3 identified computer system parameters that must be considered in smart structures and detailed cost and performance metrics that were used in this study.

The primary results of this research were obtained from a three-processor testbed network and from simulation experiments of a damage detection, location, and estimation (DLE) algorithm. From the outset, this research recognized that the effectiveness of a multicomputer implementation must account for specific details of the algorithm it is executing. The testbed results discussed in Chapter 4 demonstrate the importance of process scheduling in distributed algorithm implementations. Furthermore, the results demonstrate that these schedules must adapt to changes in the ratio of communication time to processing time.

No one topology or processing scheme can perform optimally for all applications. Chapter 5 discussed the costs associated with a multicomputer implementation of a damage DLE algorithm for a particular structure, a ten-bay truss. Chapter 6 presented simulation results for this truss. This research points out the importance of designing algorithms that take the physical characteristics of the application into account so that they can be efficiently implemented on a parallel architecture. It also demonstrates the importance of creating topologies that are suited to the traffic patterns exhibited by algorithms. An important result is that hierarchical architectures can be used in smart structures to provide acceptable performance while also limiting the cost of the network.

This final chapter examines the results obtained in previous chapters and provides overall conclusions concerning the use of multicomputers in smart structures. It examines both the cost and performance results obtained in the experiments and discusses the implications of these results for smart structures in general. The chapter ends with a discussion of future work on multicomputers for smart structures.

## *7.1. Suitability of Multicomputers for Smart Structures*

Processing systems for smart structures monitor measurements from sensors and process this information to obtain some final result about the state of the system. In health monitoring applications, this result is a report that describes the state of the structure. In control applications, the result is a sequence of control signals that drive actuators distributed throughout the structure. In either case, the sensor measurements used in the calculations are received from sites distributed throughout the structure. Thus, the sensors and actuators create input/output (I/O) requirements that are inherently distributed in space. A single centralized processor would have difficulty accommodating the I/O requirements of large structures.

Distributed processing elements that preprocess and analyze the sensor and actuator data, however, are ideally suited for such an I/O-intensive application.

Another consideration for computers in smart structure is that they should scale with the size of the structure. Centralized processing schemes have to be greatly modified if the processing load is steadily increased. In addition to the I/O problems mentioned above, the processing power of a single processing element is also a limitation. Thus, the ability of a multicomputer to scale to the size of the problem by adding processing nodes is a key characteristic that is required in smart structures.

## *7.2. Cost Issues*

The use of several interconnection network topologies to execute a specific damage DLE algorithm for a ten-bay truss was examined in Chapter 5. Through this examination, the costs of the topologies were exposed. Many of the factors leading to these costs are expected to be relevant for most applications. Thus, the metrics and methodology used in this study could be applied to other systems.

The same algorithm is used for each of the topologies, so the processing requirement in each implementation is approximately the same. The custom planar topology requires one processor for each structural element in the truss. If the size of the truss is increased, one processing element must be added for every structural element added to the truss. The custom hierarchical topology requires all of the processors that exist in the custom planar topology, plus four additional processors to form an upper level. Additional levels need to be added to the hierarchy as the size of the structure increases, but the lowest level of the hierarchy contains the majority of the processing elements. Thus, the number of processors required in the custom hierarchical topology configuration remains approximately equal to the number of processors required in the custom planar topology, even for large applications. The hypercube implementation requires one processing element for each structural element, but it also requires that the number of processors in the system be a power of two. For some cases, this may require the number of additional processing elements required to complete the network to be approximately equal to the number of processors required to perform the calculations for the truss elements. Such an additional cost may be justifiable in small systems, but it becomes prohibitive as system size increases.

The results obtained show that custom-built topologies can be implemented with lower node costs than standard topologies that require some regular structure in the array of processing elements. The results also show that the number of processors needed to form additional hierarchical layers on top of a single level implementation does not significantly increase the total number of processors required in the system.

Similar results for cost were discovered when the number of communication links in the system was examined. The regular structure of the hypercube, along with the communication links that are required to produce this regularity, cause the hypercube to need almost twice as many communication links as the custom planar topology. The custom hierarchical topology, on the other hand, requires only about 15 percent more links than the custom planar topology for the ten-bay truss.

The final cost consideration is the amount of embedded fiber needed for each topology. Once again, the hypercube requires the most embedded fiber and the custom planar topology requires the least. One reason for this is that it is difficult to efficiently map a six-dimensional hypercube onto a two-dimensional structure. This problem becomes more pronounced as larger structures are considered and hypercubes with more dimensions are required. For the custom planar topology, however, the amount of embedded fiber required scales linearly with the size of the structure. The custom hierarchical topology shows a balance between the rich connectivity and small network diameter provided by the hypercube and the ability to efficiently scale the amount of embedded fiber required as the size of the structure increases.

Another consideration concerning the amount of embedded fiber required in the network is demonstrated by the testbed. This hybrid sensing/communication network showed that the optical fibers used for the communication links can also be used as sensors. Since both fiber optic sensing and interprocessor communication are required in smart structures, it may be economical to merge the two functions on a single embedded optical fiber. The sensing mechanism used in this research is not accurate enough to be used in "real-world" applications, but it does demonstrate that wavelength division multiplexing can be used to integrate communication and sensing functions on a single fiber. Further research is needed to develop sensors that can simultaneously support communication signals and accurately monitor sensed phenomena.

### *7.3. Algorithm Performance Issues*

The design of any computer system must account for the algorithms that the system executes. Computer systems for smart structures implement control algorithms that are dominated by the manipulation and multiplication of matrix equations. Thus, the inclusion of vector processing capabilities is recommended in smart structure processing nodes. Another finding is the importance of effective techniques for message broadcasts. Many control algorithms require global matrix information to be available at all nodes in the structure. The average delay for broadcast messages in all of the simulations was much larger than the average delay of point-to-point messages. This supports conclusions in the literature about the importance of efficient broadcast techniques for parallel implementations of control algorithms.

Another factor affecting algorithm performance is the ratio of communication time to processing time. The processing time for the testbed network discussed in Chapter 4 is much larger than the communication time. In this system, the effects of message latency are negligible. Instead, process scheduling and data dependencies are the major causes of processing delay. The addition of math coprocessors changes the relative timing of the processes executed in each algorithm, and this alters the relative performance of the algorithms. For the system described in Chapter 6, the cycle time is also dominated by the processing time when the processing time is set to five times the communication time. Communication delays that are significant when the processing time and communication times are approximately equal are not an important consideration when the processing time increases. Data dependency, however, can be a cause of significant delay, regardless of the ratio of processing time to communication time.

The importance of event scheduling in cases where the communication time is significant was shown in Chapter 6. The results indicate that as background message traffic increases, the times at which non-broadcast algorithm-based messages are generated spreads out. As these messages spread out, the average and standard deviation of message latency decreases even though the total amount of traffic in the network increases. This is a clear example of how scheduling algorithm-based message traffic at appropriate times in the processing cycle can improve performance.

From these results, it is seen that the effective implementation of an algorithm on a multicomputer in a smart structure requires proper process scheduling to minimize the cycle time. Also, in systems where communication delay makes up a significant portion of the total cycle time, a further examination of the interconnection network performance is required.

### *7.4. Interconnection Network Performance Issues*

The simulations performed in this research examine three different interconnection network topologies. The hypercube topology is able to handle far more background traffic than the custom topologies. The message latency in the hypercube, however, is only marginally lower than latency in the custom hierarchical topology for cases where the background message rate is small enough such that it does not saturate the communication links in the custom hierarchical topology.

The hypercube has the greatest performance advantage when the message latency of background messages is examined. Background messages with uniformly distributed sources and destinations propagate quickly through the hypercube network, but significantly degrade the performance of the two modified-mesh topologies considered in this study. This is because the mesh topologies are tailored for the traffic patterns that are expected in the algorithm; they are not well suited for traffic with random source-destination pairs. Thus, as the amount of background traffic in the network increases, the mesh topologies saturate more readily than the richly interconnected hypercube topology. In a real application where all of the sources of message traffic are known, the topology could be designed to accommodate message traffic patterns generated by all processes executed in the application.

A significant result is that the custom hierarchical topology performs significantly better than the hypercube when non-broadcast algorithm-based messages are examined. The custom planar topology also has a smaller standard deviation of message delay than the hypercube for these messages. These results demonstrate that custom-designed topologies that account for the traffic patterns that are required for an algorithm can outperform a general-purpose topology even though the general-purpose topology may contain many more communication links and be more expensive to construct.

The examination of the three topologies leads to several conclusions concerning appropriate network choices. The algorithm examined in this study exhibits characteristics of structural locality that enable it to be easily decomposed into tasks assigned to processors distributed throughout the structure. An important feature of this locality is that it enables the topology of the interconnection network to use short links because the processors that need to communicate with each other the most are usually physically close. The amount of embedded fiber required for highly connected groups of localized processors is smaller than the amount required to connect a few distant processors. Thus, algorithms that exhibit physical locality reduce the amount of embedded fiber required in the structure.

In most implementations, there will inevitably be a need for distant processor nodes to communicate. If this communication interferes with the communication occurring on the highly connected local processors, the performance of the entire communication system is degraded. A small group of long communication links can be added to the network to decouple the performance of local communication from the performance of distant communication. This sets up a hierarchical structure in which long communication links are on a separate layer of the hierarchy than local communication links.

The custom hierarchical topology is a hierarchical structure examined in this research. As was noted previously, the performance of the custom hierarchical topology is on par with the performance of the hypercube. The cost of the custom hierarchical topology, however, is much less than the hypercube. In fact, the cost of the custom hierarchical topology is only slightly higher than the cost of the custom planar topology. This additional cost is worthwhile considering the performance increase that is obtained by the addition of the upper layer of the hierarchy.

## *7.5. Future Work*

This research presented an initial examination of the application of multicomputers to smart structures. Although several critical issues have been examined, additional work is required before multicomputers can be effectively implemented in smart structures. First, control algorithms need to be developed that exhibit the physical locality of computation that is demonstrated by the damage DLE algorithm studied in this research. This will make multicomputer implementations feasible for a larger number of applications. Also, multicomputers with hundreds or thousands of nodes should be simulated to verify how the results of the ten-bay experiments extend to larger structures. Concerns such as the message routing mechanisms, fault tolerance, and power distribution should be more thoroughly investigated. Fully integrated systems that are larger than the three-processor testbed system used in this research should be constructed and evaluated. These evaluations would lead to the construction of larger systems that will allow the requirements and capabilities of multicomputers for smart structures to be more rigorously demonstrated and examined.

# Bibliography

1. R.O. Claus, "Fiber Sensors as Nerves for 'Smart Materials'," *Photonics Spectra*, Vol. 25, No. 4, Apr. 1991, p. 75.

2. K. Talat, "Smart Skins and Fiber-Optic Sensors Applications and Issues," *SPIE Vol. 986: Fiber Optic Smart Structures and Skins III*, 1990, pp. 103-114.

3. K. Talat, "Fiber Sensors Take Wing in Smart-Skin Applications," *Photonics Spectra*, Vol. 25, No. 4, Apr. 1991, pp. 85-88.

4. R.M. Measures, "Fiber Optics Smart Structures Program at UTIAS," *SPIE Vol. 1170: Fiber Optic Smart Skins II*, 1989, pp. 93-108.

5. R.M. Measures, "Fiber Optic Sensors - The Key to Smart Structures," *SPIE Vol. 1120: Fiber Optics*, 1989, pp. 161-174.

6. R.M. Measures, "Smart Structures with Nerves of Glass," *Progress in Aerospace Sciences*, Vol. 26, No. 6, 1989, pp. 289-351.

7. C.J. Mazur, G.P. Sendeckyj, and D.M. Stevens, "Air Force Smart Structures/Skins Program Overview," *SPIE Vol. 986: Fiber Optic Smart Structures and Skins*, 1988, pp. 19-29.

8. "Coping with an Aging Fleet," *Aerospace Engineering*, Vol. 26, No. 4, Jan. 1990, pp. 13-17.

9. E. Udd, *et al.*, "Fiber-Optic Sensor Systems for Aerospace Applications," *SPIE Vol. 838: Fiber Optic Sensors V*, 1987, pp. 162-168.

10. E. Udd, "Overview of Fiber Optic Smart Structures for Aerospace Applications," *SPIE Vol. 986: Fiber Optic Smart Structures and Skins*, 1988, pp. 3-5.

11. A.M. Vengsarkar and K.A. Murphy, "Fiber Sensors in Aerospace Applications: The Smart Structures Concept," *Photonics Spectra*, Vol. 24, No. 4, Apr. 1990, pp. 119-124.

12. B.S. Thompson, M.V. Gandhi, and S. Kasiviswanathan, "An Introduction to Smart Materials and Structures," *Materials & Design*, Vol. 13, No. 1, February 1992, pp. 3-9.

13. W.J. Rowe, "Prospects for Intelligent Aerospace Structures," *AIAA/SOLE 2nd Aerospace Maintenance Conf.*, AIAA-86-1139, 1986, pp. 1-9.

14. T.L. Thomas, "Development of Robust Hybrids for Smart Skin Avionics Applications," *Proc. Electronic Components & Technology Conf.*, 1990, pp. 131-139.

15. J. Rhea, "The Next Generation of Avionics," *Air Force Magazine*, Vol. 73, No. 1, Jan. 1990, pp. 68-72.

16. J.R. Todd, "Toward Fly-by-Light Aircraft," *SPIE Vol. 989: Fiber Optic Systems for Mobile Platforms II*, 1988, pp. 38-42.

17. W.B. Scott, "Air Force Funding Joint Studies to Develop 'Smart Skin' Avionics," *Aviation Week & Space Technology*, Vol. 128, No. 16, Apr. 18, 1988, pp. 65-69.

18. C.J. Murray, " 'Smart' Structures Assess Damage in Flight," *Design News*, Vol. 45, No. 8, Apr. 24, 1989, pp. 128-129.

19. E. Udd, "Fiber Optic Smart Structures for Aerospace Applications," *European Conf. on Smart Structures and Materials*, 1992, pp. 7-12.

20. A.M. Vengsarkar, "Fiber Optic Sensors: A Comparative Evaluation," *Photonics Design and Application Handbook*, Vol. 37, No. 3, 1991, pp. H-114 - H-116.

21. J. Lesko, G. Carmen, K. Reifsnider, A. Vengsarkar, B. Miller, B. Fogg, and R. Claus, "Application of Fabry-Perot Fiber Optic Sensors in Composite Macro Models," *Proc. Conf. Optical Fiber Sensor-Based Smart Materials and Structures*, Technomic Publishing Co., 1991, pp. 65-69.

22. K.L. Belsey, J.B. Carroll, L.A. Hess, D.R. Huber, and D. Schmadel, "Optically Multiplexed Interferometric Fiber Optic Sensor System," *SPIE Vol. 566: Fiber Optic and Laser Sensors III*, 1985, pp. 257-264.

23. D.E. Cox and D.K. Lindner, "Active Control for Vibration Suppression in a Flexible Beam Using a Modal Domain Optical Fiber Sensor," *ASME Journ. of Vibration and Acoustics*, Vol. 113, No. 3, July 1991, pp. 369-382.

24. D.K. Lindner and K. Reichard, "Weighted Distributed-Effect Sensors for Smart Structure Applications," *APDA/AIAA/ASME/SPIE Conf. on Active Materials and Adaptive Structures*, 1991, pp. 53-58.

25. K.A. Murphy, B.R. Fogg, A.M. Vengsarkar, and R.O. Claus, "Spatially Weighted Fiber Optic Sensors for Smart Structure Applications," *APDA/AIAA/ASME/SPIE Conf. on Active Materials and Adaptive Structures*, 1991, pp. 43-46.

26. R.O. Claus, K.D. Bennet, A.M. Vengsarkar, and K.A. Murphy, "Embedded Optical Fiber Sensor for Materials Evaluation," *Journ. of Nondestructive Evaluation*, Vol. 8, No. 2, June 1989, pp. 135-145.

27. A. Dasgupta, Y. Wan, J.S. Sirkis, and Harmeet Singh, "Micro-mechanical Investigation of an Optical Fiber Embedded in a Laminated Composite," *SPIE Vol. 1370: Fiber Optic Smart Structures and Skins III*, 1990, pp. 119-128.

28. S.L. Ehlers, K.J. Jones, R.E. Morgan, and J. Hixson, "Composite Embedded Fiber Optic Data Links in Avionics Packaging," *Proc. Conf. Optical Fiber Sensor-Based Smart Materials and Structures*, Technomic Publishing Co., 1991, pp. 93-101.

29. C.K. Lee, "Theory of Laminated Piezoelectric Plates for the Design of Distributed Sensors/Actuators. Part I: Governing Equations and Reciprocal Relationships," *Journ. Acoust. Soc. Am.*, Vol. 87, No. 3, Mar. 1990, pp. 1144-1158.

30. N.W. Hagood, E.F. Crawley, J. de Luis, and E.H. Anderson, "Development of Integrated Components for Control of Intelligent Structures," *Smart Materials, Structures, and Mathematical Issues*, Technomic Publishing Co., 1989, pp. 80-104.

31. S. Hanagud, M.W. Obal, and A.G. Calise, "Piezoceramic Devices and PVDF Films as Sensors and Actuators for Intelligent Structures," *Smart Materials, Structures, and Mathematical Issues*, Technomic Publishing Co., 1989, pp. 69-79.

32. T. Bailey and J.E. Hubbard, Jr., "Distributed Piezoelectric-Polymer Active Vibration Control of a Cantilever Beam," *Journ. of Guidance and Control*, Vol. 8, No. 5, Sept.-Oct. 1985, pp. 605-611.

33. H.S. Tzou and J.P. Zhong, "Adaptive Piezoelectric Shell Structures: Theory and Experiments," *APDA/AIAA/ASME/SPIE Conf. on Active Materials and Adaptive Structures*, 1991, pp. 719-724.

34. D.C. Zimmerman and D.J. Inman, "On the Nature of the Interaction Between Structures and Proof-Mass Actuators," *Journ. of Guidance*, Vol. 13, No. 1, Jan.-Feb. 1990, pp. 82-88.

35. D.C. Zimmerman, G.C. Horner, and D.J. Inman, "Microprocessor Controlled Force Actuator," *Journ. of Guidance*, Vol. 11, No. 3, May-June 1988, pp. 230-236.

36. W.S. Anders, C.A. Rogers, and C.R. Fuller, "Control of Sound Radiation from Shape Memory Alloy Hybrid Composite Panels by Adaptive Alternate Resonance Tuning," *AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conf.*, Part 1 of 4, 1991, pp. 159-168.

37. W.S. Anders and C.A. Rogers, "Design of a Shape Memory Alloy Deployment Hinge for Reflector Facets," *AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conf.*, Part 1 of 4, 1991, pp. 148-158.

38. D.R. Morgenthaler, "Passive and Active Control of Space Structures: Phase I," Vol. 1, WRDC-TR-90-3044, Wright-Patterson Air Force Base, Ohio, 1990.

39. B.C. Kuo, *Automatic Control Systems: Fifth Edition*, Prentice Hall, Inc., Englewood Cliffs, New Jersey, 1987.

40. J.D. Gardiner and A.J. Laub, "Implementation of Two Control System Design Algorithms on a Message-Passing Hypercube," *Proc. 2nd Conf. on Hypercube Multiprocessors*, 1987, pp. 512-519.

41. J.D. Gardiner and A.J. Laub, "Hypercube Implementation of Some Parallel Algorithms in Control," *Advanced Computing Concepts and Techniques in Control Engineering*, Vol. F47, 1988, pp. 361-389.

42. P.R. Cappello and A.J. Laub, "Systolic Computation of Multivariable Frequency Response," *IEEE Trans. Automatic Control*, Vol. 33, No. 6, June 1988, pp. 550-558.

43. R.N. Gehling, D.R. Morgenthaler, and K.E. Richards, Jr., *Passive and Active Control of Space Structures (PACOSS): Phase II*, WL-TR-91-3052, Wright Patterson Air Force Base, Ohio, 1991.

44. B.A. Frances, *A Course in $H_\infty$ Control*," Springer-Verlag, Heidelberg, Germany, 1987.

45. K.N. Chiang and R.E. Fulton, "Concepts and Implementation of Parallel Finite Element Analysis," *Computers & Structures*, Vol. 36, No. 6, 1990, pp. 1039-1046.

46. C.K. Lee and F.C. Moon, "Modal Sensors/Actuators," *Journal of Applied Mechanics, Transactions of the ASME*, Vol. 57, No. 2, June 1990, pp. 434-441.

47. C.C. Chu and F.R. Chang, "Some Results on the Problems of Reliable Stabilization," *Int'l. Journ. of Control*, Vol. 53, No. 6, 1991, pp. 1343-1358.

48. K. Watanabe and S.G. Tzafestas, "A Hierarchical Multiple Model Adaptive Control of Discrete-Time Stochastic Systems for Sensor and Actuator Uncertainties," *Automatica*, Vol. 26, No. 5, 1990, 875-886.

49. M. Aldeen and M. Jamshidi, "Decentralized Control of Large-Scale Dynamical Systems Incorporating Feedforward Compensators," *Computers and Electrical Engineering*, Vol. 16, No. 2, 1990, pp. 99-108.

50. M.J. Balas, "Trends in Large Space Structure Control Theory: Fondest Hopes, Wildest Dreams," *IEEE Trans. Automatic Control*, Vol. AC-27, No. 3, June 1982, pp. 522-535.

51. D.K. Lindner, G. Twitty, and R. Goff, "Damage Detection, Location, and Estimation for Large Truss Structures," *Proc. AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conf.*, 1993, pp. 1539-1548.

52. R.P. Lippmann, "An Introduction to Computing with Neural Nets," *IEEE ASSP Magazine*, Vol. 4, No. 2, Apr. 1987, pp. 4-22.

53. B. Kosko, *Neural Networks and Fuzzy Systems: A Dynamical Systems Approach to Machine Intelligence*, Prentice Hall, Englewood Cliffs, NJ, 1992.

54. J.L. McClelland and D.E. Rumelhart, *Explorations in Parallel Distributed Processing: A Handbook of Models, Programs, and Exercises*, MIT Press, Cambridge, MA, 1988.

55. K.S. Narendra and K. Parthasarathy, "Identification and Control of Dynamical Systems Using Neural Networks," *IEEE Trans. on Neural Networks*, Vol. 1, No. 1, Mar. 1990, pp. 4-27.

56. M.R. Napolitano and C.I. Chen, "Application of Neural Network to the Active Control of Structural Vibration," *APDA/AIAA/ASME/SPIE Conf. on Active Materials and Adaptive Structures*, 1991, pp. 247-252.

57. M.R. Napolitano and C.I. Chen, "Numerical Investigation of the Application of a Neural Observer for State Estimation Purposes in the Active Control of Structural Vibrations for a Cantilevered Beam," *submitted to the IOP Journal*, 1991.

58. B. Grossman, H. Hou, and R. Nassar, "Optical Processors for Smart Structures," *SPIE Vol. 1296: Advances in Information Processing IV*, 1990, pp. 403-413.

59. B. Grossman, H. Hou, and R. Nassar, "Electrooptic Processors for Smart Sensors," *IEEE SOUTHCON*, 1990, pp. 314-319.

60. M. Thursby, K. Yoo, and B. Grossman, "Neural Control of Smart Electromagnetic Structures," *SPIE Vol. 1588: Fiber Optic Smart Structures and Skins IV*, 1991, pp. 219-228.

61. M.H. Thursby, B. Grossman, T. Alavie, and K.S. Yoo, "Smart Structure Incorporating Artificial Neural Networks, Fiber-Optic Sensors, and Solid-State Actuators," *Spie Vol. 1170: Fiber Optic Smart Structures and Skins II*, 1989, pp. 316-325.

62. H. Smith, Jr., "Smart Structures for Combat Aircraft," *ISA Int. Conf. & Exhibition*, Vol. 45, Part 4, 1990, pp. 1659-1668.

63. T. Williams, "Real-time Multiprocessing Pushes Software Limits," *Computer Design*, Vol. 30, No. 14, Nov. 1991.

64. J.A. Stankovic, "Misconceptions About Real-Time Computing: A Serious Problem for Next-Generation Systems," *Computer*, Vol. 21, No. 10, Oct. 1988, pp. 10-19.

65. S.M. Islam and H.H. Ammar, "Performability Analysis of Distributed Real-Time Systems," *IEEE Trans. Comp.*, Vol. 40, No. 11, Nov. 1991, pp. 1239-1251.

66. C.P. Ahrens and R.O. Claus, "A Real-Time Controller for Applications in Smart Structures," *SPIE Vol. 1170: Fiber Optic Smart Structures and Skins*, 1989, pp. 384-392.

67. K.N. Chiang and R.E. Fulton, "Structural Dynamics Methods for Concurrent Processing Computers," *Computers & Structures*, Vol. 36, No. 6, 1990, pp. 1031-1037.

68. X. Zhang, "System Effects of Interprocessor Communication Latency in Multicomputers," *IEEE Micro*, Vol. 11, No. 2, Apr. 1991, pp. 12-55.

69. R. Berrendorf and J. Helin, "Evaluating the Basic Performance of the Intel iPSC/860 Parallel Computer," *Concurrency: Practice and Experience*, Vol. 4, No. 3, May 1992, pp. 223-240.

70. G. Zorpette, "The Power of Parallelism," *IEEE Spectrum*, Vol. 29, No. 9, Sept. 1992, pp. 28-33.

71. P. Close, "The iPSC/2 Node Architecture," *Concurrent Supercomputing, The Second Generation: A Technical Summary of the iPSC/2 Concurrent Supercomputer*, Intel Corporation, Beaverton, Oregon, 1988, pp. 43-50.

72. R. Arlauskas, "iPSC/2 System: A Second Generation Hypercube," *Concurrent Supercomputing, The Second Generation: A Technical Summary of the iPSC/2 Concurrent Supercomputer*, Intel Corporation, Beaverton, Oregon, 1988, pp. 9-13.

73. P. Walker, "The Transputer," *Byte*, Vol. 10, No. 5, May 1985, pp. 219-235.

74. R. Pountain, "The Transputer Strikes Back," *Byte*, Vol. 16, No. 8, Aug. 1991, pp. 265-275.

75. D.A. Reed, D.C. Grunwald, "The performance of Multicomputer Interconnection Networks," *Computer*, Vol. 20, No. 6, June 1987, pp. 63-72.

76. D.P. Agrawal and V.K. Janakiram, "Evaluating the Performance of Multicomputer Configurations," *Computer*, Vol. 19, No. 5, May 1986, pp. 23-37.

77. D.A. Reed and R.M. Fujimoto, *Multicomputer Networks: Message-Based Parallel Processing*, MIT Press, Cambridge, Mass., 1987.

78. T.Y. Feng, "A Survey of Interconnection Networks," *Computer*, Vol. 14, No. 12, Dec. 1981, pp. 12-27.

79. L.M. Ni and P.K. McKinley, "A Survey of Wormhole Routing Techniques in Direct Networks," *Computer*, Vol. 26, No. 2, Feb. 1993, pp. 62-76.

80. W.J. Dally, "Virtual-Channel Flow Control," *IEEE Int'l Symp. Comp. Arch.*, 1990, pp. 60-68.

81. W.J. Dally, "Performance Analysis of k-ary n-cube Interconnection Networks," *IEEE Trans. Comp.*, Vol. 39, No. 6, June 1990, pp. 775-785.

82. C.J. Glass and L.M. Ni, "The Turn Model for Adaptive Routing," *Proc. Int. Symp. Comp. Architecture*, 1992, pp. 278-287.

83. L.D. Wittie, "Communication Structures for Large Networks of Microcomputers," *IEEE Trans. on Comp.*, Vol. C-30, No. 4, Apr. 1981, pp 264-273.

84. R.J. McMillen and H.J. Siegel, "Evaluation of Cube and Data Manipulator Networks," *Journ. of Parallel and Distributed Computing*, Vol. 2, No. 1, Feb. 1985, pp. 79-107.

85. H.J. Siegel, W.T.Y. Hsu, and W. Jeng, "Interconnection Networks: The Multistage Cube, Extra-Stage Cube, and Dynamic Redundancy Networks," *Proc. New Frontiers in Comp. Architecture Conf.*, 1986, pp. 1-19.

86. J. Ghosh and K. Hwang, "Critical Issues in Mapping Neural Networks on Message-Passing Multicomputers," *Proc. Int'l. Symp. Comp. Architecture*, Vol. 16, No. 2, 1988, pp. 3-11.

87. K. Hwang and J. Ghosh, "Hypernet: A Communication-Efficient Architecture for Constructing Massively Parallel Computers," *IEEE Trans. Comp.*, Vol. C-36, No. 12, Dec. 1987, pp. 1450-1466.

88. J.H. Moreno and T. Lang, "Matrix Computations on Systolic-Type Meshes," *Computer*, Vol. 23, No. 4, Apr. 1990, pp. 32-51.

89. R. Duncan, "A Survey of Parallel Computer Architectures," *Computer*, Vol. 23, No. 2, Feb. 1990, pp. 5-16.

90. J. Ghosh and K. Hwang, "Optically Connected Multiprocessors for Simulating Artificial Neural Networks," *SPIE Vol. 882: Neural Network Models for Optical Computing*, 1988, pp. 2-11.

91. X. Zhang, M. McKenna, J.P. Mesirov, and D.L. Waltz, "The Back Propagation Algorithm on Grid and Hypercube Architectures," *Parallel Computing*, Vol. 14, No. 3, Aug. 1990, pp. 317-327.

92. J.E. Steck, B.M. McMillin, K. Krishnamurthy, M.R. Ashouri, and G.G. Leininger, "Parallel Implementation of a Recursive Least Squares Neural Network Training Method on the Intel iPSC/2," *Int. Conf. on Neural Networks*, Vol. 1, 1990, pp. 631-636.

93. J.T. McHenry, "A Performance Evaluation of Multicomputer Networks for Real-Time Computing," *Masters Thesis*, Virginia Polytechnic Institute and State University, Blacksburg, VA, 1990.

94. J. Fiddler, D.N. Wilner, and H. Wong, "Multiprocessing: An Extension of Distributed, Real-Time Computing," *IEEE COMPCON*, 1990, pp. 216-218.

95. K.G. Shin, "HARTS: A Distributed Real-Time Architecture," *Computer*, Vol. 24, No. 5, May 1991, pp. 25-35.

96. E.J. O'Neil and H. Allik, "Performance of Blocked Gaussian Elimination on Multiprocessors," *Proc. SIAM Conf. Parallel Processing for Scientific Computing*, 1987, pp. 32-35.

97. G.A. Geist and C.H. Romine, "LU Factorization on Distributed-Memory Multiprocessors," *Proc. SIAM Conf. Parallel Processing for Scientific Computing*, 1987, pp. 15-18.

98. S. Chandran and L.S. Davis, "Parallel Vision Algorithms: An Approach," *Proc. SIAM Conf. Parallel Processing for Scientific Computing*, 1987, pp. 235-249.

99. R. Daniel and K. Teague, "A Connectionist Technique for Data Smoothing," *Proc. Distributed Memory Comp. Conf.*, 1990, pp. 154-157.

100. J.P. Zhu, "An Efficient FFT Algorithm on Multiprocessors with Distributed Memory," *Proc. Distributed Memory Comp. Conf.*, 1990, pp. 358-363.

101. B.K. Wada, J.I. Fanson, and E.F. Crawley, "Adaptive Structures," *Mechanical Engineering*, Vol. 112, No. 11, Nov. 1990, pp. 41-46.

102. M.R. Haskard, "An Experiment in Smart Sensor Design," *Sensors and Actuators*, Vol. 24, No. 2, July 1990, pp. 163-169.

103. A.W. van Herwaarden and R.F. Wolffenbuttel, "Introduction to Sensors Compatible with Microprocessors," *Microprocessors and Microsystems*, Vol. 14, No. 2, Mar. 1990, pp. 74-82.

104. J.T. McHenry, S.F. Midkiff, J.A. Wiencko, T.W. Reed, "Dual MIL-STD-1773 Communication and Microbend Sensor Fiber Optic Link," *Proc. Conf. Optical Fiber Sensor-Based Smart Materials and Structures*, Technomic Publishing Co., 1991, pp. 222-226.

105. J.T. McHenry and S.F. Midkiff, "Hybrid Sensing/Communication Fiber Optic Networks for Smart Structure Applications," *Journal of Smart Materials and Structures*, Vol. 1, No. 2, June 1992, pp. 146-155.

106. C. Leung, C. Huang, and I.F. Chang, "Communication-Sensing System Using a Single Optical Fiber," *Proc. SPIE Vol. 838: Fiber and Laser Sensors V*, 1987, pp. 294-298.

107. K.Y. Chen, C.Y. Leung, and I.F. Chang, "Integrated Communication and Sensing System Using One Single-Mode Optical Fibre," *Elect. Lett.*, Vol. 24, No. 13, June 23, 1988, pp. 790-792.

108. P.L. Fuhr, P.J. Kajenski, and W.B. Spillman, "Simultaneous Digital Optical Communications and Vibration Sensing on a Single Multimode Fiber," *Proc. Conf. Fiber Sensor-Based Smart Materials and Structures*, Technomic Publishing Co., 1991, pp. 201-206.

109. J.A. Wiencko, Jr., "Hybrid Sensing/Communication Systems," Fiber & Electro-Optics Research Center Internal Report, VPI&SU, Blacksburg, VA, Apr. 1989.

110. J.A. Wiencko, Jr., "Hybrid Sensing/Communication Systems for Smart Structures: A White Paper on Suggested Research Directions of Research for Hybrid Systems," Fiber & Electro-Optics Research Center, VPI&SU, Blacksburg, VA, Dec. 1989.

111. A.D. Kersey, A. Dandridge, and A.B. Tveten, "Overview of Multiplexing Techniques for Interferometric Fiber Sensors," *SPIE Vol. 838: Fiber Optic and Laser Sensors V*, 1987, pp. 184-193.

112. R.E. Morgan, S.L. Ehlers, and K.J. Jones, "Composite-Embedded Fiber Optic Data Links and Related Material/Connector Issues," *APDA/AIAA/ASME/SPIE Conf. on Active Materials and Adaptive Structures*, 1991, pp. 775-779.

113. F.P. Beer and E.R. Johnston, Jr., *Mechanics of Materials*, second edition, McGraw-Hill, Inc., New York, 1992.

114. "MIL-STD-1773: Fiber Optics Mechanization of an Aircraft Internal Time-Division Command/Response Multiplex Data Bus," Naval Air Engineering Center, Systems Engineering and Standardization Department, Lakehurst, NJ, 1988.

115. *MIL-STD-1553 Designers Guide*, ILC Data Device Corporation, Bohemia, NY, 1988.

116. K.S. Trivedi, *Probability & Statistics with Reliability, Queuing, and Computer Science Applications*, Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1982.

117. S.H. Bokhari, "On the Mapping Problem," *IEEE Trans. Comp.*, Vol. C-30, No. 3, Mar. 1981, pp. 207-214.

118. D.L. Perry, *VHDL*, McGraw-Hill, Inc., New York, NY, 1991.

119. J.R. Armstrong, *Chip-Level Modeling with VHDL*, Prentice Hall, Inc., Englewood Cliffs, NJ, 1989.

# Appendix A. Confidence Intervals for Figure Data

Table A.1. Confidence intervals for data in Figure 6.3a

Hypercube

| Gen. Rate | Plotted Point | Lower Bound | Upper Bound |
|---|---|---|---|
| 0.00 | 12.85 | 12.85 | 12.85 |
| 0.05 | 13.34 | 12.67 | 14.01 |
| 0.10 | 13.99 | 13.47 | 14.51 |
| 0.20 | 15.14 | 14.14 | 16.14 |
| 1.00 | 21.80 | 19.81 | 23.78 |

2-D mesh

| Gen. Rate | Plotted Point | Lower Bound | Upper Bound |
|---|---|---|---|
| 0.00 | 22.84 | 22.84 | 22.84 |
| 0.05 | 25.01 | 23.99 | 26.03 |
| 0.10 | 27.97 | 26.20 | 29.74 |

Pyramid mesh

| Gen. Rate | Plotted Point | Lower Bound | Upper Bound |
|---|---|---|---|
| 0.00 | 12.87 | 12.87 | 12.87 |
| 0.05 | 13.70 | 13.61 | 13.78 |
| 0.10 | 14.57 | 13.29 | 15.84 |
| 0.20 | 16.22 | 16.07 | 16.36 |

Table A.2. Confidence intervals for data in Figure 6.3b

Hypercube

| Gen. Rate | Plotted Point | Lower Bound | Upper Bound |
|---|---|---|---|
| 0.00 | 23.54 | 23.54 | 23.54 |
| 0.05 | 23.95 | 23.49 | 24.41 |
| 0.10 | 24.54 | 24.38 | 24.70 |
| 0.20 | 25.69 | 25.24 | 26.14 |
| 1.00 | 31.85 | 30.59 | 33.10 |

2-D mesh

| Gen. Rate | Plotted Point | Lower Bound | Upper Bound |
|---|---|---|---|
| 0.00 | 33.52 | 33.52 | 33.52 |
| 0.05 | 35.93 | 35.15 | 36.71 |
| 0.10 | 38.70 | 36.63 | 40.77 |

Pyramid mesh

| Gen. Rate | Plotted Point | Lower Bound | Upper Bound |
|---|---|---|---|
| 0.00 | 23.56 | 23.56 | 23.56 |
| 0.05 | 24.43 | 24.12 | 24.73 |
| 0.10 | 24.87 | 24.42 | 26.36 |
| 0.20 | 26.65 | 26.17 | 27.12 |

Table A.3. Confidence intervals for data in Figure 6.3c

Hypercube

| Gen. Rate | Plotted Point | Lower Bound | Upper Bound |
|---|---|---|---|
| 0.00 | 77.00 | 77.00 | 77.00 |
| 0.05 | 77.46 | 77.02 | 77.90 |
| 0.10 | 78.21 | 77.78 | 78.65 |
| 0.20 | 79.08 | 78.86 | 79.30 |
| 1.00 | 85.16 | 84.58 | 85.74 |

2-D mesh

| Gen. Rate | Plotted Point | Lower Bound | Upper Bound |
|---|---|---|---|
| 0.00 | 86.99 | 86.99 | 86.99 |
| 0.05 | 88.99 | 87.05 | 90.93 |
| 0.10 | 92.25 | 90.08 | 94.42 |

Pyramid mesh

| Gen. Rate | Plotted Point | Lower Bound | Upper Bound |
|---|---|---|---|
| 0.00 | 77.02 | 77.02 | 77.02 |
| 0.05 | 77.79 | 77.56 | 78.02 |
| 0.10 | 78.98 | 78.51 | 79.44 |
| 0.20 | 80.55 | 79.87 | 81.23 |

Table A.4. Confidence intervals for data in Figure 6.4

Hypercube

| Gen. Rate | Plotted Point | Lower Bound | Upper Bound |
|---|---|---|---|
| 0.00 | 3.13 | 3.13 | 3.13 |
| 0.05 | 2.74 | 2.64 | 2.83 |
| 0.10 | 2.45 | 2.40 | 2.48 |
| 0.20 | 2.05 | 2.02 | 2.13 |
| 1.00 | 1.29 | 1.24 | 1.33 |

2-D mesh

| Gen. Rate | Plotted Point | Lower Bound | Upper Bound |
|---|---|---|---|
| 0.00 | 6.33 | 6.33 | 6.33 |
| 0.05 | 5.88 | 5.70 | 6.06 |
| 0.10 | 5.59 | 5.11 | 6.06 |

Pyramid mesh

| Gen. Rate | Plotted Point | Lower Bound | Upper Bound |
|---|---|---|---|
| 0.00 | 3.60 | 3.60 | 3.60 |
| 0.05 | 3.07 | 2.99 | 3.14 |
| 0.10 | 2.79 | 2.77 | 2.81 |
| 0.20 | 2.34 | 2.15 | 2.52 |

Table A.5. Confidence intervals for data in Figure 6.5

Hypercube

| Gen. Rate | Plotted Point | Lower Bound | Upper Bound |
|---|---|---|---|
| 0.00 | 4.00 | 4.00 | 4.00 |
| 0.05 | 4.13 | 3.96 | 4.29 |
| 0.10 | 4.22 | 4.18 | 4.26 |
| 0.20 | 4.50 | 4.32 | 4.67 |
| 1.00 | 6.02 | 5.80 | 6.24 |

2-D mesh

| Gen. Rate | Plotted Point | Lower Bound | Upper Bound |
|---|---|---|---|
| 0.00 | 6.89 | 6.89 | 6.89 |
| 0.05 | 7.79 | 7.57 | 8.01 |
| 0.10 | 8.71 | 7.82 | 9.60 |

Pyramid mesh

| Gen. Rate | Plotted Point | Lower Bound | Upper Bound |
|---|---|---|---|
| 0.00 | 3.66 | 3.66 | 3.66 |
| 0.05 | 3.78 | 3.78 | 3.78 |
| 0.10 | 3.99 | 3.93 | 4.05 |
| 0.20 | 4.24 | 3.70 | 4.78 |

Table A.6. Confidence intervals for data in Figure 6.6

Hypercube

| Gen. Rate | Plotted Point | Lower Bound | Upper Bound |
|---|---|---|---|
| 0.00 | ---- | ---- | ---- |
| 0.05 | 0.137 | 0.133 | 0.142 |
| 0.10 | 0.141 | 0.135 | 0.147 |
| 0.20 | 0.160 | 0.154 | 0.165 |
| 1.00 | 0.356 | 0.339 | 0.374 |

2-D mesh

| Gen. Rate | Plotted Point | Lower Bound | Upper Bound |
|---|---|---|---|
| 0.00 | ---- | ---- | ---- |
| 0.05 | 0.242 | 0.214 | 0.269 |
| 0.10 | 0.258 | 0.226 | 0.290 |

Pyramid mesh

| Gen. Rate | Plotted Point | Lower Bound | Upper Bound |
|---|---|---|---|
| 0.00 | ---- | ---- | ---- |
| 0.05 | 0.174 | 0.133 | 0.214 |
| 0.10 | 0.188 | 0.184 | 0.192 |
| 0.20 | 0.239 | 0.236 | 0.241 |

Table A.7. Confidence intervals for data in Figure 6.7

Hypercube

| Gen. Rate | Plotted Point | Lower Bound | Upper Bound |
|---|---|---|---|
| 0.00 | ---- | ---- | ---- |
| 0.05 | 0.159 | 0.110 | 0.208 |
| 0.10 | 0.174 | 0.065 | 0.283 |
| 0.20 | 0.244 | 0.191 | 0.297 |
| 1.00 | 0.687 | 0.673 | 0.701 |

2-D mesh

| Gen. Rate | Plotted Point | Lower Bound | Upper Bound |
|---|---|---|---|
| 0.00 | ---- | ---- | ---- |
| 0.05 | 0.701 | 0.514 | 0.887 |
| 0.10 | 0.649 | 0.391 | 0.907 |

Pyramid mesh

| Gen. Rate | Plotted Point | Lower Bound | Upper Bound |
|---|---|---|---|
| 0.00 | ---- | ---- | ---- |
| 0.05 | 0.302 | 0.053 | 0.550 |
| 0.10 | 0.330 | 0.324 | 0.336 |
| 0.20 | 0.463 | 0.416 | 0.510 |

Table A.8. Confidence intervals for data in Figure 6.8

Hypercube

| Gen. Rate | Plotted Point | Lower Bound | Upper Bound |
|---|---|---|---|
| 0.00 | 0.125 | 0.125 | 0.125 |
| 0.05 | 0.128 | 0.127 | 0.130 |
| 0.10 | 0.120 | 0.109 | 0.131 |
| 0.20 | 0.129 | 0.117 | 0.141 |
| 1.00 | 0.219 | 0.213 | 0.225 |

2-D mesh

| Gen. Rate | Plotted Point | Lower Bound | Upper Bound |
|---|---|---|---|
| 0.00 | 0.124 | 0.124 | 0.124 |
| 0.05 | 0.135 | 0.125 | 0.146 |
| 0.10 | 0.135 | 0.122 | 0.148 |

Pyramid mesh

| Gen. Rate | Plotted Point | Lower Bound | Upper Bound |
|---|---|---|---|
| 0.00 | 0.108 | 0.108 | 0.108 |
| 0.05 | 0.100 | 0.095 | 0.106 |
| 0.10 | 0.106 | 0.079 | 0.132 |
| 0.20 | 0.103 | 0.101 | 0.104 |

Table A.9. Confidence intervals for data in Figure 6.9

Hypercube

| Gen. Rate | Plotted Point | Lower Bound | Upper Bound |
|---|---|---|---|
| 0.00 | 0.348 | 0.348 | 0.348 |
| 0.05 | 0.355 | 0.332 | 0.378 |
| 0.10 | 0.352 | 0.335 | 0.370 |
| 0.20 | 0.372 | 0.347 | 0.397 |
| 1.00 | 0.480 | 0.453 | 0.507 |

2-D mesh

| Gen. Rate | Plotted Point | Lower Bound | Upper Bound |
|---|---|---|---|
| 0.00 | 0.111 | 0.111 | 0.111 |
| 0.05 | 0.136 | 0.106 | 0.166 |
| 0.10 | 0.139 | 0.118 | 0.159 |

Pyramid mesh

| Gen. Rate | Plotted Point | Lower Bound | Upper Bound |
|---|---|---|---|
| 0.00 | 0.051 | 0.051 | 0.051 |
| 0.05 | 0.055 | 0.030 | 0.080 |
| 0.10 | 0.080 | 0.00 | 0.162 |
| 0.20 | 0.073 | 0.058 | 0.087 |

# Vita

John T. McHenry

June 1993

## Education

**Aug 1990 - Jun 1993** *Virginia Polytechnic Institute and State University, Blacksburg, VA.* Ph.D., Electrical Engineering.

**Aug 1988 - May 1990** *Virginia Polytechnic Institute and State University, Blacksburg, VA.* M.S., Electrical Engineering. Thesis: "Performance Evaluation of Multicomputer Networks for Real-Time Computing."

**Sep 1983 - Jun 1988** *Virginia Polytechnic Institute and State University, Blacksburg, VA.* B.S., Electrical Engineering. Summa Cum Laude.

## Work Experience

| | |
|---|---|
| Department of Defense | Cooperative Education |
| Fort Meade, Maryland | March 1985 - Present |

Two summer internships with the Supercomputing Research Center.
Application development for the SPLASH I systolic array processor.
Designed and tested interface circuitry for the SPLASH 2 systolic array processor.
Designed, tested, and implemented digital communication interface circuitry and driver software.
Assembled and tested a secure telephone switching unit.

## Honors/Awards

Virginia Tech's Bradley Graduate Fellowship: 1988-1993.
Virginia Tech's Outstanding EE Cooperative Education Senior Award: 1988.
Three special achievement cash awards from the Department of Defense.
Dean's list every academic term.
President of the Virginia Tech Racquetball Club: 1991-1992.

Publications

J.T. McHenry and S.F. Midkiff, "Hybrid Sensing/Communication Fiber Optic Multicomputer Networks for Smart Structure Applications," *Journal of Smart Materials and Structures*, Vol. 1, No. 2, 1992, pp. 146-155.

S.F. Midkiff and J.T. McHenry, "Multicomputer Networks for Smart Structures," *ADPA/AIAA/ASME/SPIE International Symposium & Exhibition on Active Materials & Adaptive Structures*, 1991, pp. 239-242.

J.T. McHenry, S.F. Midkiff, J.A. Wiencko, and T.W. Reed, "Dual MIL-STD-1773 Communication and Microbend Sensor Fiber Optic Link," *Proc. Optical Fiber Sensor-Based Smart Materials and Structures*, Technomic Publishing Co., 1991, pp. 222-226.

J.T. McHenry, S.F. Midkiff, and N.J. Davis, "Performance Evaluation of Multicomputer Networks for Real-Time Computing," *Distributed Memory Computing Conference*, 1990, pp. 1314-1323.