N-37
306
p. 59

NASA Contractor Report 194467

Parallel Computing for Probabilistic Response Analysis of High Temperature Composites

R.H. Sues, Y.J. Lua, and M.D. Smith
Applied Research Associates, Inc.
Albuquerque, New Mexico

March 1994

Prepared for
Lewis Research Center
Under Contract NAS3–26576

(NASA-CR-194467) PARALLEL COMPUTING FOR PROBABILISTIC RESPONSE ANALYSIS OF HIGH TEMPERATURE COMPOSITES Final Report (Applied Research Associates) 59 p

N94-27656

Unclas

G3/39 0000306

National Aeronautics and
Space Administration

PARALLEL COMPUTING FOR PROBABILISTIC RESPONSE ANALYSIS OF HIGH TEMPERATURE COMPOSITES

Prepared by:

R. H. Sues
Y. J. Lua
M. D. Smith

Applied Research Associates, Inc.
4300 San Mateo Blvd., NE, Suite A220
Albuquerque, New Mexico 87110

TABLE OF CONTENTS

Section	Title	Page
1.0	**INTRODUCTION**	1-1
1.1	Background	1-1
1.2	Objectives and Scope	1-1
1.3	Parallel Processing Background	1-2
2.0	**COMPUTATIONAL STRATEGIES FOR PARALLEL PROBABILISTIC COMPOSITE MECHANICS**	2-1
2.1	Introduction	2-1
2.2	Brief Review of Deterministic Composite Mechanics	2-2
2.2.1	Introduction	2-2
2.2.2	Micromechanics Approach to Composite Materials	2-3
2.3	Probabilistic Composite Mechanics	2-5
2.4	Parallelism in PCM and Exploitation Strategies	2-6
2.4.1	Introduction	2-6
2.4.2	Parallelism in Probabilistic Response of a Composite Laminate	2-7
2.4.3	Comprehensive Strategy for Parallel PCM	2-9
3.0	**PARALLEL IMPLEMENTATION ON SHARED- AND DISTRIBUTED-MEMORY ARCHITECTURES**	3-1
3.1	Introduction	3-1
3.2	Shared-Memory Architecture	3-1
3.2.1	Alliant FX/2800 Shared-Memory Architecture	3-1
3.2.2	Physical Shared-Memory Programming Paradigm	3-2
3.2.3	3D Space Truss	3-4
3.2.3.1	Problem Description and Probabilistic Analysis Results	3-4
3.2.3.2	Parallel Performance	3-6
3.2.4	Fatigue Crack Growth	3-10
3.2.4.1	Problem Description and Probabilistic Analysis Results	3-10
3.2.4.2	Parallel Performance	3-14
3.3	Distributed-Memory Architecture	3-16
3.3.1	Message-Passing Programming Paradigm with CS Tools	3-16
3.3.2	Virtual Shared-Memory Programming Paradigm with C-Linda	3-20
3.3.2.1	Overview	3-20
3.2.2.2	Application of C-Linda for Multi-Level Parallelism	3-22
3.3.3	Parallel Performance of the Fatigue Reliability Problem with C-Linda	3-28
4.0	**SUMMARY, CONCLUSIONS, AND RECOMMENDATIONS**	4-1
4.1	Summary	4-1
4.2	Conclusions and Recommendations	4-2
5.0	**REFERENCES**	5-1

LIST OF TABLES

Table	Title	Page
2-1	Sources of Parallelism in Various PSM Methods	2-6
3-1	Material Property Random Variables for 3D Space Truss	3-5
3-2	Loading Random Variables for 3D Space Truss	3-5
3-3	Effect of Average Bandwidth on Parallel Efficiency for 3D Space Truss on Alliant FX/2800 (4/Board Configuration)	3-9
3-4	Fatigue Life Reliability Problem Random Variables	3-11
3-5	Statistical Parameters of the Fatigue Life	3-13
3-6	Basic Linda Functions	3-23

LIST OF FIGURES

Figure	Title	Page
1-1	Parallel Architectures — Memory Taxonomy	1-4
2-1	A Five-Ply Laminate	2-7
2-2	Lamina with Woven Fibers	2-8
2-3	Finite Element Model of a Representative Volume Element for a Woven Lamina	2-8
2-4	Computational Strategy for Parallel PCM	2-10
3-1	Alliant FX/2800 Architecture	3-2
3-2	3D Space Truss Example — Front Panel	3-4
3-3	CDF for Stress in the Rear Vertical Element at the Base of the 3D Space Truss	3-5
3-4	Alliant FX/2800 Processor Allocation Configurations	3-7
3-5	Parallel Speedup for 3D Space Truss on Alliant FX-2800	3-7
3-6	Parallel Efficiency for 3D Space Truss on Alliant FX/2000	3-8
3-7	Variation of Parallel Speedup with Number of Simulation Trials	3-9
3-8	Single-Edged Fatigue Crack Growth Example	3-11
3-9	PDF of Initial Crack Length	3-12
3-10	Fatigue Life CDF	3-13
3-11	Fatigue Crack Paths from Random Initial Orientation	3-14
3-12	Parallel Efficiency for Fatigue Reliability Problems on Alliant FX/2800	3-15
3-13	Parallel Speedup for Fatigue Reliability Problem on Alliant FX/2800	3-16
3-14	Distributed-Memory Software Development: Message-Passing Paradigm with CS Tools	3-17
3-15	Master-Slave Model for Distributed-Memory System	3-18
3-16	Implementation of Master-Slave Model for Parallel Fatigue Reliability Analysis with CS Tools	3-19

3-17 Message Passing 3-22

3-18 Live Data Structures 3-22

3-19 Distributed Data Structures 3-22

3-20 Distributed Memory Software Development: Virtual Shared-Memory Paradigm with C-Linda 3-23

3-21 Parallel Speedup for Fatigue Reliability Problem on SPARCStation Network 3-29

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

Advanced composite materials are playing an increasingly important role in aerospace structures and spacecraft. However, new high temperature composites will be needed to answer the challenges posed by 21st century aeropropulsion structures. These high performance lightweight structures will be subjected to a variety of complex, severe cyclic and transient thermal and mechanical loading environments. Research into methods for analyzing the response of these structures has resulted in analytical tools that can be used to tailor their design [*e.g.*, Chamis, 1986a] and reduce the need for costly experiments. These analysis methods combine the micro- and macro-mechanical aspects of computational composite mechanics.

The need to incorporate uncertainties in the inputs and structure modeling for both analysis and optimum design has also been recognized [Chamis, 1986b; Stock, *et al.*, 1988; Chamis and Stock 1990; Thanedar & Chamis, 1990; Mase, *et al.*, 1991]. Uncertainties arise at both the micro- and macro-mechanical levels. For example, uncertainties arise due to the scatter of the constituent (fiber and matrix) properties, and fabrication process variables. In general, it is necessary to consider uncertainties in loadings, material properties, boundary conditions, geometry, and system response and failure modeling.

While the aforementioned analytical methods are effective tools, they can require considerable computer time. They typically entail finite element analysis of large nonlinear problems with analysis of progressive failure that requires load stepping and/or time stepping solutions. A single deterministic problem can strain available resources, hence, the repeated analyses required for probabilistic simulations and optimal design can be severely constrained.

Fortunately many sources of parallelism are inherent in probabilistic composite mechanics (PCM) problems. Thus, these problems are ideally suited to computation on parallel processing computers. However, the software strategies to achieve large-scale parallelism across a range of parallel architectures do not exist. To achieve large-scale parallelism it will be necessary to judiciously exploit the multiple levels of parallelism in PCM problems. Inappropriate parallelization can actually result in reduction of speedup with increasing numbers of processors. Developments in specially adapted computational strategies, along with the rapid advances occurring in massively parallel hardware, will significantly speed application of probabilistic composite analysis and design methods. Use of these methods to tailor composites and meet reliability-based design criteria will contribute to making application of high temperature composites in aerospace propulsion possible.

1.2 OBJECTIVES AND SCOPE

The overall goal of this research program is to achieve large-scale parallelism in solving problems in probabilistic response analysis of high temperature composites. To do this we must

be able to keep large numbers of processors busy with a minimum of parallel overhead. In addition, since this research is part of the Small Business Innovative Research program, potential commercialization of the research is important, and we have adopted a goal to develop a hardware/software package that is marketable and meets the following requirements: (1) the software/hardware package should be available for low-end to high-end price ranges (*e.g.*, be able to operate on networks of workstations to massively parallel supercomputers), (2) we should not require special purpose hardware, and (3) the software should be portable, extensible, and able to adapt as hardware advances are made. This report presents the results of the Phase I research to determine the feasibility of developing such a system. The following specific Phase I objectives can be enumerated:

1. Identify the multiple levels of parallelism in probabilistic response analysis of high-temperature composites and identify the innovative computational strategies needed to exploit this parallelism.

2. Evaluate the efficiency of two parallel computing architectures through example problem applications.

3. Formulate recommendations for optimal hardware configurations for particular classes of problems, and formulate optimal software strategies for the different hardware configurations and problems.

To meet these objectives we conducted a number of investigations. These results are summarized in the following two chapters. In Chapter 2 we present the results of a review of the phases of PCM, identify the multiple levels of parallelism, and discuss strategies for exploiting this parallelism. In Chapter 3 we present the results of several software implementation and hardware efficiency investigations that were undertaken to establish the most promising approaches to follow in future research and ultimately for commercialization. The software implementations included a physical shared-memory model (for shared-memory computers), a message passing model (for distributed-memory computers), and a virtual shared-memory model (for either architecture, for hybrid architectures, and for networks of workstations). The hardware implementations included a shared-memory computer with twenty-four processors, and a distributed-memory network of seven workstations. Two example problems were executed on the shared-memory computer (3D Space Truss analysis and a fatigue life reliability analysis) and one example on the distributed-memory network (fatigue life reliability). In Chapter 4 we present our conclusions and recommendations. The next section of this chapter provides some background on parallel processing that will be useful in reading the remainder of this report.

1.3 PARALLEL PROCESSING BACKGROUND

The purpose of parallel processing is to improve the speed with which a computational task can be done by breaking it into subtasks and executing as many as possible of these subtasks simultaneously. This idea actually has a long history in computer science, but has received greater attention recently with the advent of affordable parallel computers. Dramatically reduced costs of components, such as high speed 64-bit RISC processors, development of efficient

interprocessor communication strategies, and reduced costs of memory are making development of parallel machines with 1000's of 64-bit processors a reality.

Our purpose here is to define some concepts that are relevant to the research reported herein. A more detailed summary of the principle ideas in parallel processing can be found in Sues, *et al.* [1991a, b]. A survey of the range of parallel architectures currently available is given by Dongarra and Duff [1992].

The two principal means to instill parallelism into a computer architecture are pipelining and concurrency. Pipelining refers to the processing of data in an assembly line fashion, the concept now widely employed in vector processing computers. Concurrency refers to the simultaneous operation of multiple independent processors. Both concepts are of importance in parallel implementations of PCM problems. That is, in parallel PCM we want to use multiple independent processors to perform independent (or pseudo-independent) tasks; but since these tasks often involve vector operations, each processor should ideally be a vector processor. Vectorization of the numerical operations in computational mechanics has been studied extensively and is well understood. It is the use of concurrency that is the focus of this research. It is concurrency that we wish to exploit to keep large numbers of processors (preferably vector processors) busy to achieve large-scale parallelism and massive reductions in computation time for PCM problems.

There have been several attempts to classify computer architectures, or create a taxonomy for them, but the field is sufficiently dynamic that new architectures which defy existing classifications continue to be created. A commonly accepted scheme is that of Flynn [1966]:

- SISD — single instruction stream, single data stream.
- SIMD — single instruction stream, multiple data stream.
- MISD — multiple instruction stream, single data stream.
- MIMD — multiple instruction stream, multiple data stream.

With regard to the preceding discussion of pipelining (vectorization) and concurrency, vector processing is typified by the SIMD model and concurrency (for our purposes) by the MIMD model. Again we note that computers exist that incorporate both architectures, that is many computers have multiple processors (MIMD), but each processor uses pipelining (SIMD) for vectorization.

For concurrent processing it is important to make the distinction between shared-memory and distributed-memory. These different architectures are depicted in Figure 1-1, which also shows some currently available commercial hardware implementations. Simply stated, shared-memory machines are composed of multiple processors that are all connected to a central (global) shared memory; whereas in a distributed memory machine, each processor has its own local memory, and the local memories are connected via a network. Various means of connecting processors to a shared-memory and various topologies for distributed-memory networks are used in practice. While shared-memory provides the fastest way for processors to

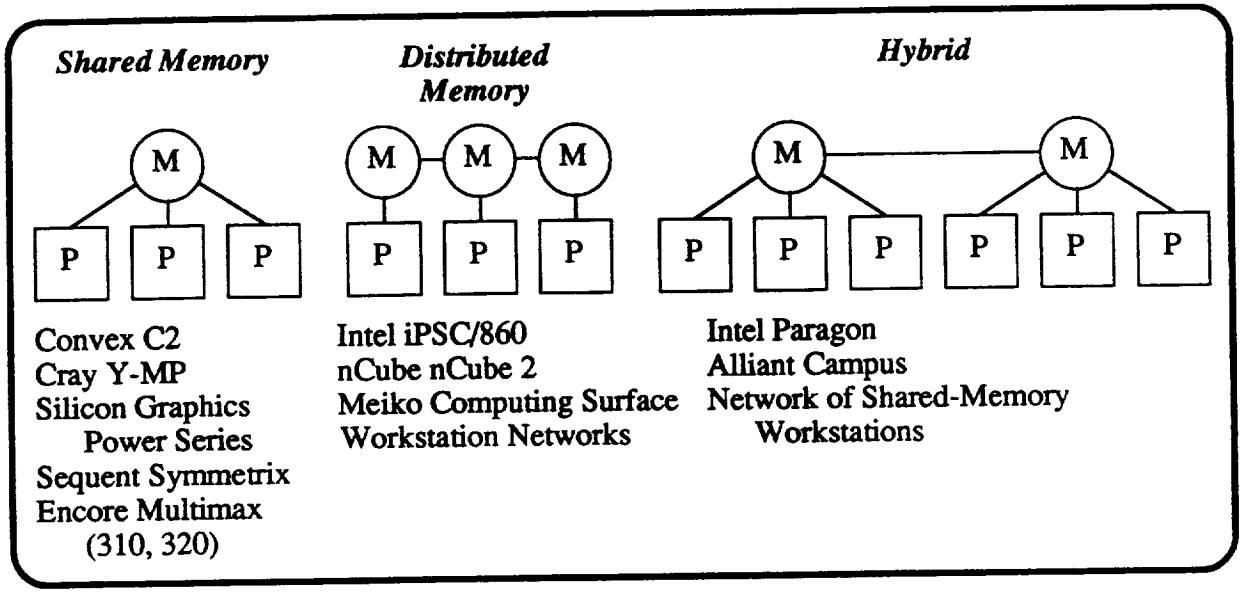


Figure 1-1. Parallel Architectures — Memory Taxonomy.

communicate with each other, the shared memory is also a source of congestion that, in practice, limits the number of processors. In contrast, distributed-memory machines are already available with large numbers of processors. The potential drawback here is the need for communication among the processors. Hybrid machines that combine both shared and distributed memory are also being developed, although these are at the early stages of commercialization. Hybrid machines are an exciting development with regard to parallelizing PCM problems because PCM problems exhibit multiple levels of parallelism that map naturally onto this architecture.

The reason for parallel processing is increased performance; hence we need a measure of efficiency in order to gauge the relative worth of alternative architectures and algorithms. For concurrent processing a simple, useful model of speedup is given by

$$S_N = \frac{1}{\alpha + (1-\alpha)/N + f(N)} \tag{1-1}$$

where α is the fraction of the code that cannot be processed in parallel, N is the number of processors, and $f(N)$ is the overhead (a function of the number of processors). Parallel efficiency can be defined as

$$\varepsilon = \frac{S_{N,obs}}{S^{T}} \times 100 \tag{1-2}$$

where $S_{N,obs}$ is the actual observed speedup and S^T is the theoretical maximum speedup obtained when $f(N) = 0$ (for $f(N) = 0$, Equation 1-1 reduces to Amdahl's law [Amdahl, 1967]). We will use these definitions for evaluating performance later in this report.

CHAPTER 2

COMPUTATIONAL STRATEGIES FOR PARALLEL PROBABILISTIC COMPOSITE MECHANICS

2.1 INTRODUCTION

PCM problems have several levels of inherent and derived parallelism. Briefly, the top level parallelism in PCM problems results from the repeated independent problem solutions required by essentially all probabilistic analysis methods (*e.g.*, independent trials in Monte Carlo Simulation (MCS), sensitivity computations in FORM/SORM, perturbation computations in response surface, independent computations required to treat multiple performance functions in FORM/SORM etc.). Lower levels derive from dividing the structure into parts and analyzing these parts using multiple processors.

In order to achieve large-scale parallelism in PCM it will be necessary to judiciously exploit these multiple levels of parallelism. Inappropriate parallelization can actually result in reduction of speedup with increasing numbers of processors. In this introduction we discuss the need to exploit multiple levels of parallelism. In the remainder of this chapter we provide first a brief review of mechanics of composites in order to better understand and identify the parallelism in analysis of composite materials, and then identify computational strategies to exploit this parallelism.

The most obvious reason to exploit multiple levels of parallelism in PCM is to increase the degree of parallelism in the problem (*i.e.*, the number of subtasks that can be performed concurrently). In this way we can keep as many processors busy as possible. This will be important for small sample MCS methods or for the non-MCS probabilistic analysis methods, and for deterministic analyses.

A less obvious, but just as important, reason to exploit multiple levels of parallelism in PCM pertains to memory demand. For solving PCM problems in parallel, if we only exploit the top level parallelism (see above), the total memory required is roughly proportional to the number of processors used. Computational mechanics problems typically have large memory requirements. On a shared-memory machine, the amount of memory is fixed and does not scale with the number of processors. Hence, as the number of processors used increases, the memory demand can exceed the physical memory available, resulting in the need to use secondary storage (*i.e.*, disk). Forcing the use of secondary storage can result in reduction of speedup with increasing numbers of processors. Thus, lower levels of parallelism can be used in addition to the top level parallelism to divide the structure into smaller parts and analyze these parts using multiple processors. For example, on a shared-memory machine with twenty-four processors wherein it is determined that there is sufficient physical memory to replicate the entire structure six times, we would break the structure into four parts and perform six independent probabilistic analysis computations concurrently.

For a distributed-memory machine, memory scales with the number of processors so we do not have the same problem as above. However, if the structure representation (*i.e.*, matrices) cannot be fit within the local memory, it can be advantageous (and for some machines mandatory) to again break the structure down into parts that fit within the local memory at the processor so that, again, secondary storage does not need to be used.

A drawback to multi-level parallelism, however, is that as more levels are exploited, they tend to become finer and finer grained. That is, the amount of computation that is performed by a processor before it must communicate results to other processors and/or receive new inputs from other processors becomes smaller and smaller. Thus, parallel efficiency will decrease. Hence, it is always advantageous to exploit the coarsest grained parts of the problem first and then exploit successively finer grained levels. For probabilistic analysis, the top level parallelism is the coarsest grained part of the problem. In order to effectively exploit the multi-level parallelism in PCM and achieve large-scale speedup, a top down approach, along with innovative computational strategies that minimize memory requirements, is needed. A strategy for parallel PCM is outlined at the conclusion of this chapter.

2.2 BRIEF REVIEW OF DETERMINISTIC COMPOSITE MECHANICS

2.2.1 Introduction

Composite materials consist of two or more different materials that form regions large enough to be regarded as continua. Composites are typically studied from two points of view: micromechanics and macromechanics.

In the macromechanics approach, the composite material is treated as a homogeneous anisotropic material. The structural component being analyzed is assumed large enough so that the effects of the constituent materials are detected only as averaged apparent properties of the composite. The material parameters used in an anisotropic material model are usually determined from simple tests such as tension, compression and simple shear. The macromechanics approach is extremely useful in global structural analysis and to verify a micromechanical model.

The main objective of the micromechanics approach is to predict the material properties of a composite from the properties of each constituent and its volume fraction. The interaction of the constituent materials is examined on a microscopic scale. Due to the freedom in choosing constituent materials, volume fractions, and stacking order and arrangement, this approach can be used to tailor a composite material to meet special design requirements (stiffness and strength). Also, the uncertainty in the processing, fabrication and manufacture operations can be incorporated in the micromechanics approach to quantify the statistical distribution of the material property of a composite. Since the construction of the global material property of a composite can be divided into independent subtasks in the micromechanics approach, it is ideally suited for parallel computation.

For the purpose of identifying parallelism in characterizing material properties of a composite and providing an efficient computational tool for tailoring high-temperature structural composites, a brief review of the micromechanics approach is presented next.

2.2.2 Micromechanics Approach to Composite Materials

The micromechanics approach is based on the Representative Volume Element (RVE). The RVE should be large enough to contain a sufficient number of material phases, (*i.e.*, it should be large compared to the scale of microstructure), but still be small compared to the entire body. A RVE would retain and represent the properties of the composite medium, which are insensitive to values of homogeneous boundary conditions. RVE provides a valuable boundary between continuum theories and microscopic theories.

Under conditions of an imposed macroscopically homogeneous stress or deformation field on the RVE given by

$$U_i = \varepsilon_{ij}^0 \, x_j, T_i = \sigma_{ij}^0 \, n_j \tag{2-1}$$

where ε_{ij}^0 and σ_{ij}^0 are constant strain and stress, respectively, and n_j is the normal direction to the component boundary, the average stress and strain are respectively defined by

$$\overline{\sigma}_{ij} = \frac{1}{V} \int_v \sigma_{ij}(\mathbf{x}) \, \mathrm{d}V \tag{2-2}$$

$$\overline{\varepsilon}_{ij} = \frac{1}{V} \int_v \varepsilon_{ij}(\mathbf{x}) \, \mathrm{d}V \tag{2-3}$$

where V is the volume of the RVE. The effective properties of a composite are defined by

$$\overline{\sigma}_{ij} = C_{ijkl}^* \overline{\varepsilon}_{kl} \, ; \; \overline{\varepsilon}_{ij} = S_{ijkl}^* \overline{\sigma}_{kl} \tag{2-4}$$

where C_{ijkl}^* are effective elastic moduli and S_{ijkl}^* are effective elastic compliances, connected by the usual reciprocity relation.

Based on the average theorem of virtual work [Hashin, 1972], an alternative definition of effective physical properties can be given in terms of strain energy and stress energy as shown below:

$$U^\varepsilon = \frac{1}{2} C_{ijkl}^* \overline{\varepsilon}_{ij} \overline{\varepsilon}_{kl} V = W^\varepsilon V \tag{2-5}$$

$$U^\sigma = \frac{1}{2} S_{ijkl}^* \overline{\sigma}_{ij} \overline{\sigma}_{kl} V = W^\sigma V \tag{2-6}$$

where U^ε is strain energy, and U^σ is stress energy.

The main objective of the micromechanics approach is to determine the effective elastic moduli C^*_{ijkl} or effective compliance S^*_{ijkl} in terms of the correspondence of the constituent materials and their volume fraction. Due to the complexity in determining stress and strain fields in a multi-phase system, various approximate techniques have been used. Two approaches have been used extensively in deriving the effective properties of a composite, namely, the strength of materials approach and the solid mechanics approach with various levels of mathematical sophistication. An excellent review on the analysis of composite materials has been given by Hashin [1983].

The solid mechanics approach has been employed for both dilute and finite concentrations of second phase materials (fibers, spherical particles, fibrous and platelet reinforcements). For a dilute concentration of inclusions, Eshelby's solution [1957] has been used to determine the effective bulk and shear modulus for spherical inclusion composites [Hashin, 1959] and for short fibers and platelet composites [Christensen, 1979]. A solution for rigid spheres embedded in the incompressible elastic matrix has been resolved by Batchelor and Green [1972].

Two of the most commonly used models for the case of finite concentration of inclusions are those of the composite spheres model [Hashin, 1962] and the self-consistent scheme [Budiansky, 1965; Wu, 1966]. The generalized self-consistent scheme based on the three-phase model has been proposed by Christensen and Lo [1979, 1986] for the determination of the effective shear modulus. The main problem of the self-consistent scheme is that it takes enormous liberties with the geometry of the material combination. In other words, the geometry is successively rearranged to view the phase under consideration as an inclusion, even if the phase is completely continuous. A problem with the composite spheres model is that it cannot provide reasonable results for systems containing single size inclusion at high concentration. Other models based on the differential scheme [Norris, 1985], the Mori-Tanaka theory [Taya and Arsenault, 1989; Benveniste, 1987], and the Eshelby equivalent inclusion method [Mura, 1982] have also been proposed recently to quantify the effective material properties of a multiphase media.

The solid mechanics approach, based on the theory of micromechanics, normally gives the expressions for the effective properties of a composite in a complicated form. Furthermore, the properties predicted by various micromechanics models have a large scatter [Chamis and Sendeckyj, 1968].

The earliest and simplest strength of materials approach was based on Voigt [1910] and Reuss [1929] approximations. Voigt analyzed the multi-phase system by assuming uniform strain in all the phases, and Reuss by assuming uniform stress in all phases. The deficiency of the Voigt model is that it introduces non-equilibrium tractions at the phase boundaries, while the Reuss model results in strain incompatibility at the phase boundaries. Hill [1952] proved that the actual overall moduli lie somewhere in the interval between that given by the Reuss and Voigt models. Thus, the Voigt and Reuss models provide the upper and lower bounds of the true effective elastic moduli.

The strength of materials approach, along with empirical factors, determined from experimental data was proposed by Halpin and Tsai [1967] for fiber composites at low fiber volume fractions. A unified set of micromechanics equations was developed by Chamis [1983] for predicting unidirectional ply geometric, mechanical, thermal, and hygral properties. This set of equations provides a useful tool for characterizing any transversely isotropic composite (a lamina).

Laminated composites have been used extensively in automobiles, aircraft and space structures. A laminate is two or more laminae bonded together to act as an integral structural element. Due to different fiber orientations in different lamina (or ply), the composite laminate can be designed to resist load in several directions. The material properties of a laminate are obtained from the properties of the constituent laminae by lamination theory. The effective properties of a lamina, which is a flat arrangement of unidirectional fibers in a matrix, can be generated from the micromechanics model as discussed before.

The two-phase composite, lamina, has a structure in which the fibers are arranged in a periodic manner thus forming a periodic array. This periodic structure allows the analysis of a lamina using a single cell. The single cell acts as a building block such that the continuum can be constructed by repeated use of this element. The relations between the average stress and strain can be obtained from the previous micromechanics analysis. The final effective moduli of a lamina can thus be determined.

From the effective properties of each lamina, lamination theory can be applied to determine the mechanical properties of the laminate. Existing lamination theories can be classified into three types: (1) displacement-based theories (classical thin plate theory [Reissner and Stavsky, 1961; Srinivas and Rao, 1970], first order shear deformation theory [Yang, *et al.*, 1966; Whitney and Pagano, 1970; Bert, 1984], and higher order theories [Whitney and Sun, 1973; Reddy, 1984]); (2) discrete laminate theories [Seide, 1980; Murakami, 1986]; and (3) stress-based theories [Pagano and Soni, 1983; Green and Naghdi, 1982].

2.3 PROBABILISTIC COMPOSITE MECHANICS

The deterministic characterization of effective properties of a composite has been discussed. However, due to the inherent heterogeneity of the composite material and uncertainty in the fabrication process, large scatter in the physical properties of a composite has been observed from experiments. For a laminated fiber-reinforced composite, three levels of uncertainty may exist. They are: (1) constituent level (fiber & matrix properties); (2) ply level (fiber volume ratio, void volume ratio, etc.); and (3) composite level (ply angle and lay-up) [Mase, *et al.*, 1991].

Recent research has begun to develop probabilistic composite mechanics methods to consider these uncertainties in the design and analysis of composite structures. Chamis and Stock [1990] have used Monte Carlo simulation and micromechanics methods to evaluate the uncertainties in unidirectional fiber composite strengths from uncertainties in the constituent

properties. For damage and failure analysis, a stochastic damage model for the rupture prediction of a multi-phase material has been developed recently by Lua, *et al.* [1992a, 1992b], and the effect of a microdefect on the fatigue life and crack path has also been studied most recently by using the stochastic boundary integral element method [Lua, *et al.*, 1992c, 1992d]. The probabilistic finite element method has been used by Engelstad and Reddy [1991] for the analysis of laminated composite shells. In this work the first-order second-moment method was employed to compute the mean and variance of the response. The unique set of micromechanics equations, coupled with the Fast Probability Integration (FPI) method, has been used by Mase, *et. al.* [1991] to determine the cumulative distribution function for selected laminate properties. The MCS method was also used by Mase, *et al.*, 1991] to evaluate the accuracy of FPI. For design, Thanedar and Chamis [1990] describe how to tailor the design of laminated composites with probabilistic constraints. Miki, *et al.* [1992] used the advanced first-order second-moment method (AFOSM) for optimum design of a composite subject to reliability-based constraints.

2.4 PARALLELISM IN PCM AND EXPLOITATION STRATEGIES

2.4.1 Introduction

From the review of probabilistic composite mechanics, we can identify three main sources of parallelism: (1) parallelism in the general probabilistic computations; (2) parallelism in the general structural mechanics computations; and (3) specialized parallelism in the probabilistic composite mechanics analysis. Parallelism in the general probabilistic computations has been discussed in detail by Sues, *et al.* [1991a, b]. This work showed that the degree of parallelism depends on the particular probabilistic method employed. Table 2-1 summarizes the parallelism in several probabilistic methods. Parallelism in the general structural mechanics computations has also been reviewed by Sues, *et al.* [1991a, b]. Farhat [1992] provides a recent review of methods of parallelization for general finite element applications. Hence, we focus here on the parallelism in composite mechanics, in particular, characterizing a composite laminate and specialized parallelism in PCM.

TABLE 2-1. SOURCES OF PARALLELISM IN VARIOUS PSM METHODS.

Method	Repeated Performance Function Evaluations for Perturbed Inputs	Multiple CDF Values	Multiple Failure Mode Analysis	Different Structural Response Locations of Interest
FORM/SORM	X	X	X	X
Direct Monte Carlo	X		X[1]	
Monte Carlo with Variance Reduction	X	X	X	X
Hybrid	X	X	X	X

[1] Only when different analysis model or method is used for different failure modes.

2.4.2 Parallelism in Probabilistic Response of a Composite Laminate

To analyze the response and predict the failure mechanism of a structure made of a laminated composite material, the material properties of the laminate have to be determined. Knowledge of how to predict the properties of the laminate from its constituents is also essential in tailoring the laminate to meet particular structural requirements. To demonstrate the inherent parallelism in determining laminate properties, a five-ply laminate is considered here (see Figure 2-1). The laminate is made of a stack of five lamina with various orientations. A typical lamina consists of a regular arrangement of fibers in a matrix. A typical lamina with woven fibers is shown in Figure 2-2.

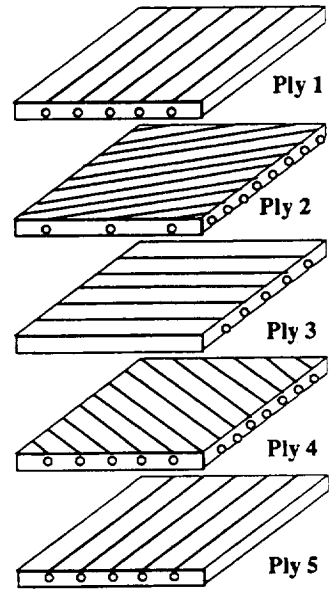


Figure 2-1. A Five-Ply Laminate.

In probabilistic analysis of the laminate properties, the properties of the fiber and matrix, the orientation of the fiber and volume fraction of each constituent (fiber or matrix) are basic random variables. For a particular simulation trial or sensitivity analysis, the ply properties need to be generated individually and independently for each realization of these basic random variables (unless the material properties of the different plies are identically distributed and perfectly correlated). For the lamina with unidirectional fibers and linear constituent behavior, the unique set of micromechanics equations [Chamis, 1983] can be used to generate the material properties for each ply. While this computation can be performed in parallel, it is relatively fine-grained due to the analytic form of the micromechanics equations.

For the case of nonlinear constitutive behavior of the constituent (fiber or matrix) or for a complex arrangement of fibers in a matrix (see Figure 2-2), finite element analysis has to be performed on a representative cell to determine the overall behavior of the lamina. Most fiber-reinforced materials with polymeric matrices exhibit nonlinear response due to viscoelastic effects. Environmental conditions, such as temperature and humidity, accelerate the viscoelastic process. Another kind of nonlinear behavior commonly exists in metal matrix lamina due to plastic deformation in the metal matrix. In addition to the nonlinear material behavior, the complex arrangement of fibers in the matrix may make the direct application of the micromechanics equations infeasible. For example, to account for the effect of curved fibers in the lamina with woven fibers (see Figure 2-2), the finite element model, shown in Figure 2-3, for a representative cell has been used. Similarly, finite element analysis of a unit cell has been performed to obtain the thermo-mechanical properties of a braided composite [Frank, *et al.*, 1991].

Finite element methods have also been used to determine the effective elastic properties for square or hexagonal periodic arrays of identical circular fibers. The computational model is

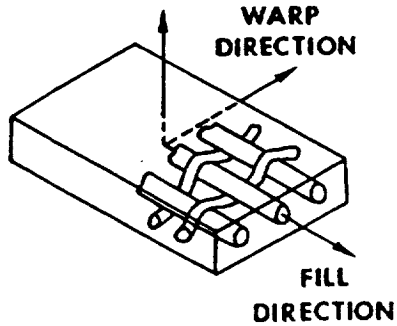


Figure 2-2. Lamina with Woven Fibers [Jones, 1975].

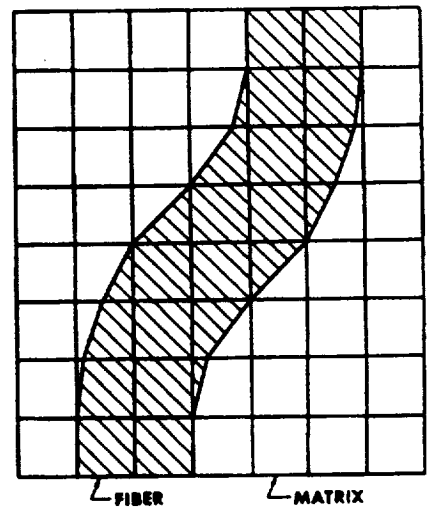


Figure 2-3. Finite Element Model of a Representative Volume Element for a Woven Lamina [Jones, 1975].

based on the typical repeating element of the array, with the boundary conditions determined by symmetry conditions. The displacement and traction continuity conditions at the interfaces of the element, as well as at the interfaces between neighboring elements, are imposed in conjunction with the equilibrium conditions. The final analysis leads to relations between the average stress and strain from which the requested effective properties are determined.

Due to the versatility of the finite element method, non-homogeneity and nonlinearity of the material can be incorporated along with various interface boundary conditions. Finite element analysis can be performed in parallel for each ply to generate the ply properties. As the FEM is computationally intensive, especially for nonlinear problems (where ply properties may need to be updated during the analysis), the problem is fairly coarse-grained and high parallel efficiency can be expected. In addition, in many cases the thermal and mechanical properties can be obtained independently and thus in parallel.

With the properties of each ply determined, the mechanical properties of a laminate can be constructed using lamination theory. Various lamination theories have been given in Section 2.2. Since the local and interfacial behavior of a laminate is critical in determining the local damage and delamination, the discrete laminate theories [Seide, 1980; Murakami, 1986] can be employed to provide an accurate analysis. In this approach, each lamina is treated as a homogeneous anisotropic plate whose properties are determined from the lamina constituent materials properties. The governing equations of all these plates are coupled through interlaminar continuity equations. To achieve greater accuracy each lamina can be divided into a number of layers. Since different parts of the structure can have different laminate properties, due to spatial variation of material properties and due to spatial variation in damage as loading progresses, independent laminate (and ply) property computations will need to be performed for different parts of the structure. These computations can be performed in parallel.

2.4.3 Comprehensive Strategy for Parallel PCM

Based on the foregoing analyses we can outline a comprehensive strategy for parallel solution of PCM problems. The approach is based on a top-down decomposition that exploits the coarsest grained part of the problem first and moves to additional levels of finer granularity as necessary. Thus, we first exploit the parallelism that is inherent in the probabilistic computations (see Table 2-1). As indicated at the beginning of this chapter the number of additional levels of parallelism that are exploited depends on several factors: (1) the number of independent problem solutions required by the probabilistic analysis, (2) the memory required for each solution, (3) the number of available processors, (4) the memory/processor, and (5) the memory configuration and/or communications architecture/bandwidth. Development of an automated control algorithm to invoke the optimum number of additional levels of parallelism based on these parameters is a proposed research task for Phase II and is discussed in more detail in the Phase II proposal.[1]

The computational strategy for parallelizing the additional levels of parallelism in PCM combines a "divide and conquer" domain decomposition strategy with two innovative techniques, probabilistic substructuring and the SPCG (stochastic preconditioned conjugate gradient) equation solution procedure [Sues, *et al.*, 1992]. Using these techniques, the overall approach is inherently parallel. The approach also minimizes memory requirements. The overall approach is depicted in Figure 2-4 for an example turbine blade analysis model and consists of three major decompositions: (1) probabilistic substructuring; (2) structural domain decomposition; and (3) composite material decomposition. The SPCG solver is used to solve the systems of equations in parallel without actually forming the global stiffness matrix.

Probabilistic Substructuring. Probabilistic substructuring is used prior to execution of the probabilistic computations in order to reduce the memory/processor requirements (a key to achieving parallel efficiency) and to reduce the execution time of each problem solution required in the subsequent probabilistic analysis. A boundary element method analog to probabilistic substructuring will be evaluated in the example problem calculation (see Chapter 3).

The probabilistic substructuring technique is illustrated in Figure 2-4b. The figure is an idealization that depicts a characteristic of many thermo-mechanical analysis problems. That is, there are regions that require detailed modeling ("hot spots") and regions that can be modeled in a coarse fashion. The regions requiring detailed modeling correspond to regions of high stress or thermal gradients, resulting from geometric discontinuities (holes, bends, intersections, etc.) or applied loads (mechanical or thermal). These regions are likely locations for initiation of failure, such as crack initiation and, for layered composites, interlaminar delamination. For probabilistic analysis, the regions requiring detailed modeling will also require more detailed treatment of uncertainties and, therefore, considerably more computational effort in both the probabilistic and thermo-mechanical aspects of the problem.

---

[1]Due to the limited scope of this Phase I feasibility study, parts of the computational strategy were tested and implemented herein (see Chapter 3); specifically, top level parallelism, probabilistic substructuring, and coding of two levels of parallelism. Fully integrated implementation is proposed for Phase II.

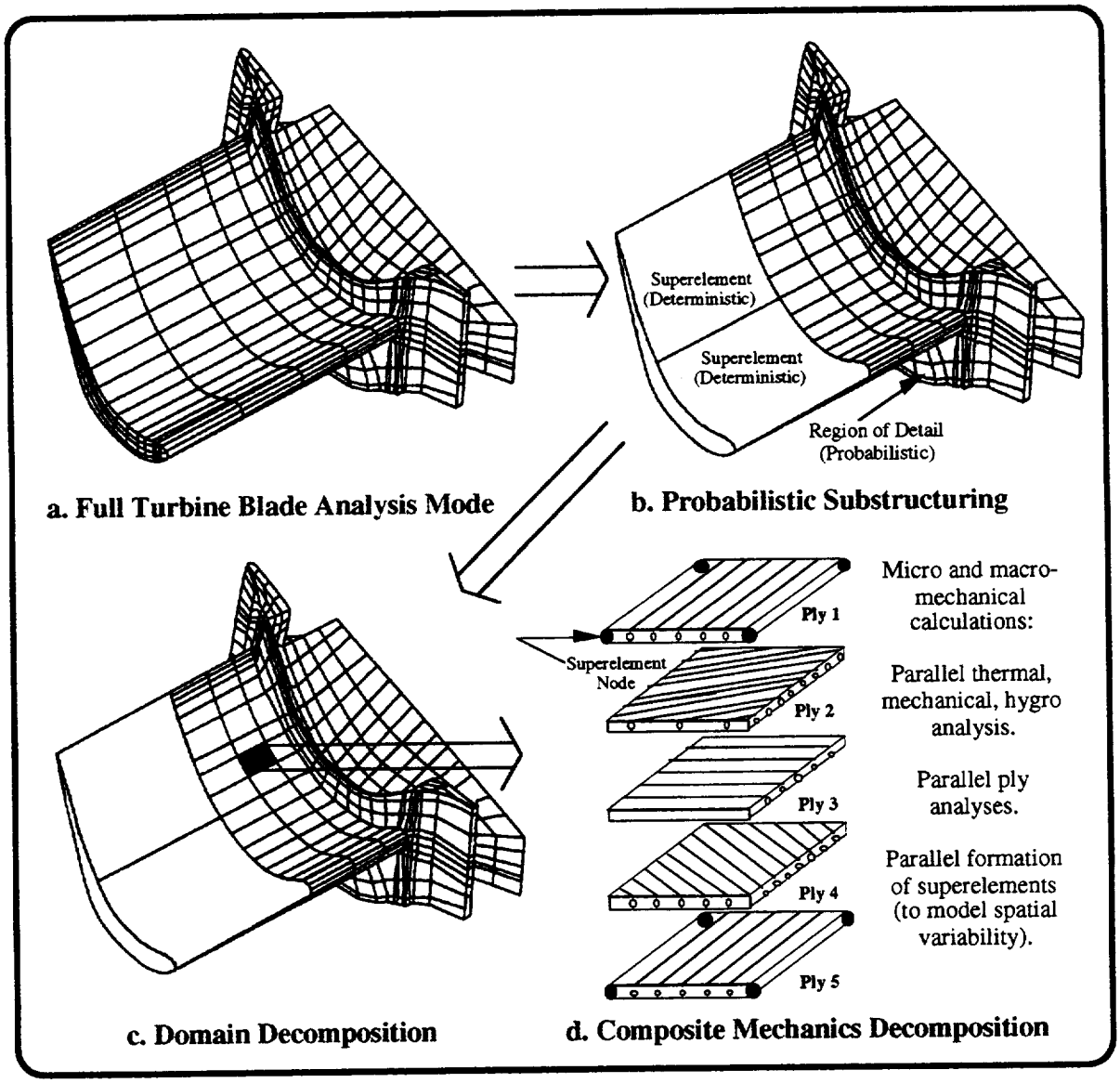


Figure 2-4. Computational Strategy for Parallel PCM.

For parallel implementation, multiple processors are first used to develop each of the superelements (we have shown two here for clarity; in practice more would be used, depending on the structure, load, and materials). This is accomplished by assigning one processor per superelement. Once the superelements are formed, the probabilistic analysis of the entire structure, which now has a much reduced number of degrees of freedom, proceeds. The structural properties of the superelement are treated deterministically; however loadings on the superelement can still be treated as random. The key contribution of this approach is in greatly reducing the memory requirements of the probabilistic analysis that, as will be shown in Chapter 3, can have a profound impact on the parallel efficiency.

Domain Decomposition. Structural domain decomposition is next used to break the structure down into distinct physical domains (illustrated by the redlines in Figure 2-4c). Each domain is assigned to a processor; thus, each processor carries out the following computations (including the composite macro and micro-mechanical computations for the domain) independent of similar computations for the other domains and hence in parallel:

1. Micro- and macro-mechanics computations for the subdomain.
2. Superelement computations for the subdomain.
3. Formation of element matrices.
4. Assembly of global matrices (for the subdomain).
5. Partial factorization of the stiffness matrix.
6. State determination or evaluation of the generalized stresses.

Composite Material Decomposition. The next level of parallelism is at the composite material micro and macro-mechanical level. This level represents a finer grain parallelism than the prior parallel sources and is invoked when processors are available and communications bandwidth allows speedups to be obtained. At this level multiple processors are used to perform micro and macro-mechanical material property computations (*i.e.*, initial element material property values and element material property updates during the structural computations) as described in Section 2.4.2.

Parallel Stochastic Preconditioned Conjugate Gradient Solver (SPCG). The stochastic preconditioned conjugate gradient solver is a very effective procedure for solving the systems of equations in probabilistic finite element analysis [Sues, *et al.*, 1992]. This solver is also ideally suited for parallel implementation since it is an iterative approach that requires only matrix-vector multiplications and vector dot products. In addition, the solver has been shown to be efficient for use with sparse schemes so that memory requirements can be minimized for large 3D problems.

CHAPTER 3

PARALLEL IMPLEMENTATION ON SHARED- AND DISTRIBUTED-MEMORY ARCHITECTURES

3.1 INTRODUCTION

Several software implementations and hardware efficiency investigations were undertaken to establish the most promising approaches to follow in Phase II and ultimately Phase III commercialization. The software implementations included a physical shared-memory model (for shared-memory computers), a message passing model (for distributed-memory computers), and a virtual shared-memory model (for either architecture, for hybrid architectures, and for networks of workstations). The hardware implementations included a shared-memory computer with twenty-four processors, and a distributed-memory network of seven workstations. The following summarizes these investigations and the purpose for each.

1. ***Shared Memory, Alliant FX/2800.*** Two problems, one with small memory requirements (a 3D space truss) and one with larger memory requirements (fatigue life reliability of a plate with an initial defect via the stochastic boundary element method). The purpose here was to investigate the affect of memory requirements on parallel efficiency for shared memory systems. Parallel speedup studies were performed using one to twenty-four processors.

2. ***Distributed Memory Software Development using Message Passing Paradigm.*** Code was developed for solving the fatigue life reliability problem using CS Tools to investigate the feasibility of parallelizing probabilistic analysis using the message passing paradigm.

3. ***Distributed Memory Software Development using Virtual-Shared Memory Paradigm.*** Code was developed for solving the fatigue life reliability problem using C-Linda to investigate the feasibility of parallelizing multiple levels of parallelism using the virtual shared-memory paradigm. Both the Monte-Carlo simulation loop and computation of the influence coefficients during each simulation history were parallelized.

4. ***Distributed-Memory Network.*** The feasibility of achieving parallel speedup on a distributed-memory network of workstations using the virtual shared-memory programming paradigm was investigated. Parallel speedup studies were performed for the fatigue reliability problem using one to seven workstations.

3.2 SHARED-MEMORY ARCHITECTURE

3.2.1 Alliant FX/2800 Shared-Memory Architecture

Figure 3-1 shows the architecture of the Alliant FX/2800. As shown, six processor modules containing four 64-*bit* Intel i860 processors, are connected with the global cache through the crossbar switch with a bandwidth of 1.28 *GB/Sec*. Each processor module has two I/O channels to the global cache, each having a bandwidth of 80 *MB/Sec*. The memory size for

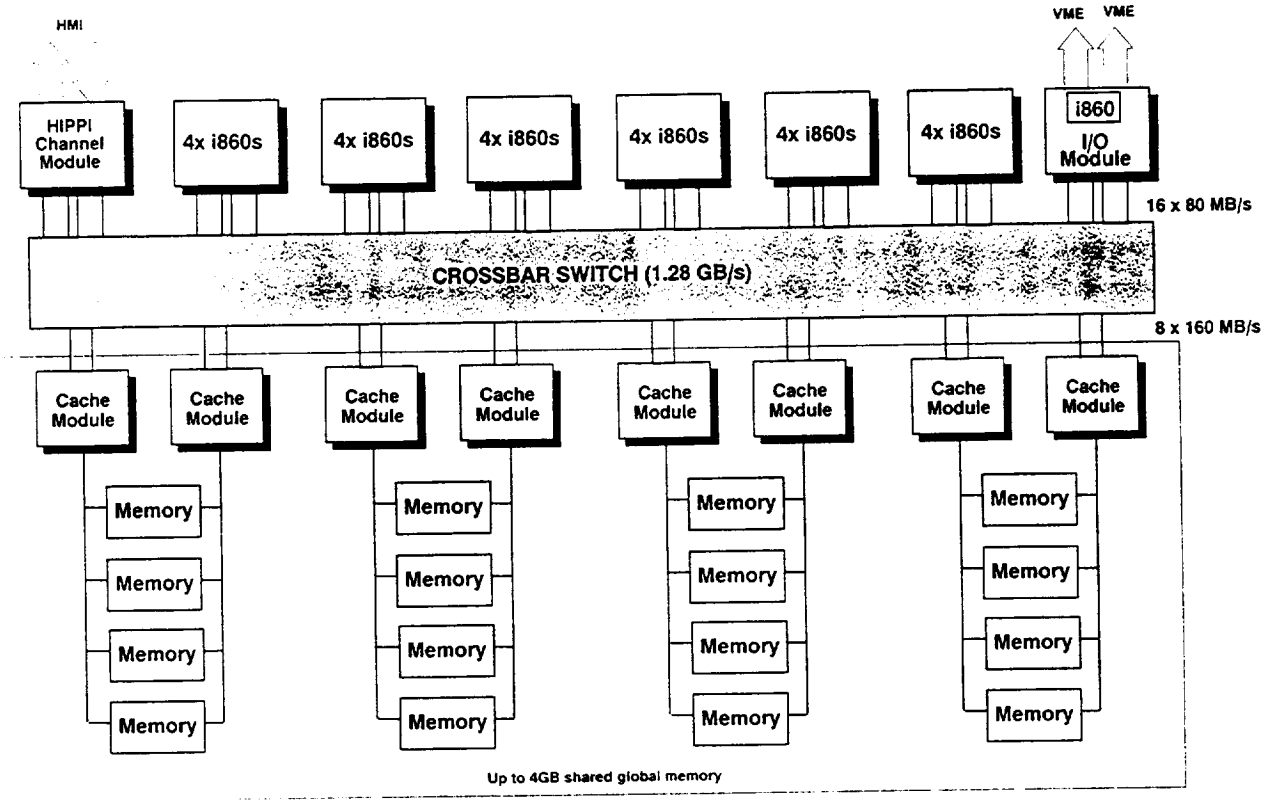


Figure 3-1. Alliant FX/2800 Architecture

each cache module is 500 *KB*; eight modules give a total of 4 *MB*. The cache modules connect to the main-memory with a total bandwidth of 640 *MB/Sec* (which is half the bandwidth from the process modules to global cache). Each memory module has a size of 64 *MB* and sixteen modules give a total of 1 *GB*.

3.2.2 Physical Shared-Memory Programming Paradigm

Parallelism in a computer program can be divided into three types, namely: (1) job-level parallelism; (2) sub-program ("task" or "macro") parallelism; and (3) loop-level ("micro") parallelism. Due to the special FORTRAN compiler on the Alliant, the micro parallelism is invoked automatically provided that no dependencies exist within the loop or between iterations (on other shared-memory computers special directives can be required). In addition to the concurrent execution of the loop, additional levels of micro parallelism can be added within the loop by using optimization directives, such as "ASSOC" (Optimize associative transformations), or "Vector" (optimize for vectorization) to maximize efficiency. To achieve macro parallelism or task parallelism, sub-tasks are grouped into recursive subroutines. As a characteristic of the recursive subroutine, a unique copy of all the local variables within the recursive subroutine is created for each concurrent process.

For MCS all subtasks that are replicated during each simulation (sampling, performance function evaluation, and scoring) are grouped into a recursive subroutine. This subroutine can then be executed concurrently since a unique copy of all local variables within the subroutine is created. An analogous approach can be used for other probabilistic methods, such as FORM/SORM, where sampling is replaced by variable perturbations at the current design point estimate. To enable parallel implementation, however, we must first use special compiler directives and coding strategies to remove dependencies between each call to the recursive subroutine. Two types of data dependencies exist in MCS. They are: (1) pseudo-random number generation in which the *i-th* random number of the sequence, $x(i)$ depends on $x(i - 1)$, and (2) scoring of the simulation results.

First, since the compiler will not automatically optimize a loop containing a subroutine call (because of the possibility for data dependency between loop iterations), optimization directives embedded in the code can be used to change the default optimization actions. Thus, by preceding the loop with a concurrent call directive and declaring the subroutine to be recursive, the concurrent execution of the loop can be achieved as shown below:

```
        Program main
CVD$ CNCALL
        do i=1,n
                call sub(a,b,c)
        end do
                ...
        end
        recursive subroutine sub(a,b,c)
                x=f(a,b,c)
        end
```

where CVD$CNCALL is the concurrent call directive for the Alliant, a, b, c are global (shared) variables, and x is a local variable. For parallel implementation of MCS, all random variables and related parameters must be defined as local variables so that a unique copy of these variables will be maintained for each concurrently executing subprogram. Conversely, deterministic problem variables and related parameters can be passed through the subroutine argument list or maintained in a common block, to be shared by all concurrently executing processes in order to minimize memory requirements and maximize computational efficiency.

To remove the data dependencies in parallel random number generation and scoring, a memory sectioning approach is employed to treat the shared-memory machine as a pseudo distributed-memory machine. By logically partitioning the shared memory into sub-memories and assigning each sub-memory a number that corresponds to a processor number, both random number generation and scoring can be performed independently within a processor's own logically-local memory. For example, in the parallel random number generation, a set of initial random seeds are generated for each processor and stored in its own logically-local memory. On each invocation of simulation trial, a function call to the Alliant's system library function LIB_PROCESSOR_NUMBER is first executed which returns the processor number of the processor that is allocated to the particular trial. The processor number is used to locate the

address of its own memory and fetch the previous set of random numbers from the memory to generate the next set of random numbers. By doing so, in the concurrent process of the simulation trials, each processor generate its own data stream and scoring results. The results stored in each local memory (*i.e.*, the section of the shared memory) will be scored after the simulation is completed. In order to maximize the problem granularity by using all available processors to execute simulations on all the processors concurrently, automatic compiler concurrency is suppressed within each simulation.

3.2.3 3D Space Truss

3.2.3.1 Problem Description and Probabilistic Analysis Results

The first example considered is a MCS analysis of a 3D space truss composed of 99 members with 72 degrees of freedom. The purpose of choosing this simple structural mechanics problem is to explore the performance of the shared-memory machine for a small problem. For the present 3D truss problem, the problem can fit completely in the global cache without using the main memory (Figure 3-1). This problem also served as a benchmark since it was studied in earlier work by Sues, *et al.* [1991a, b] on an Alliant FX/80 with eight processors.

Figure 3-2 shows the front panel and section details of the truss. The truss is made up of three panels so that the cross section is an equilateral triangle. It is simply supported at three points on the bottom and is capped by a pyramid at the top. The apex of the pyramid is circumferentially constrained so that only vertical movement is possible. Three loads are applied to the structure, a vertical load at the apex and two horizontal loads. The problem random variables are: the member elastic moduli, the member cross sectional areas, the initial strain in the members and the loadings S1, S2, and S3. Statistical descriptions of the random variables are given in Tables 3-1 and 3-2. A sample size, Nsim = 10,000, is used in the MCS. For probabilistic analysis, the performance function was defined to obtain the cumulative distribution function (CDF) for the stress in the rear vertical element at the base of the truss. The CDF is shown in Figure 3-3.

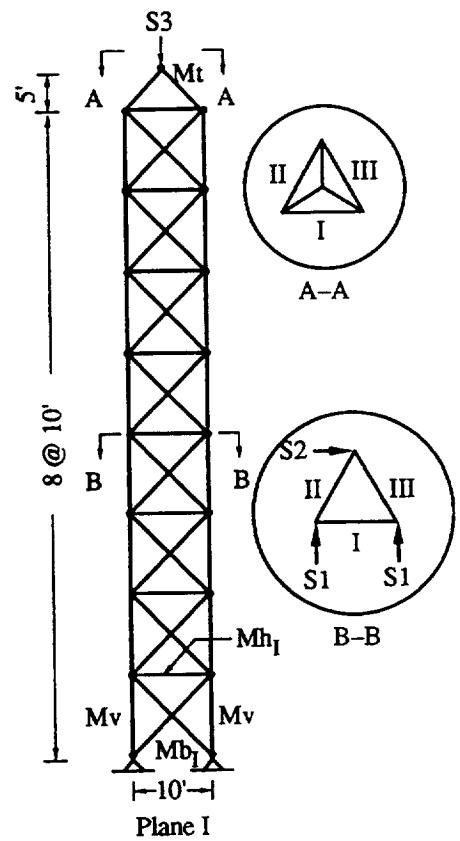


Figure 3-2. 3D Space Truss Example — Front Panel.

TABLE 3-1. MATERIAL PROPERTY RANDOM VARIABLES FOR 3D SPACE TRUSS.

Material Type	E (δ=0.10) (*ksi*)	A (δ=0.10) (sq. *in*)	ε_{int} (σ=10^{-4})
Mv	29000.	1.590	0.
Mh_I	29000.	1.590	0.
Mh_{II}	29000.	1.590	0.
Mh_{III}	29000.	1.590	0.
Mb_I	29000.	0.938	0.
Mb_{II}	29000.	0.938	0.
Mb_{III}	29000.	0.938	0.
Mt	29000.	1.590	0.

E = mean modulus of elasticity (lognormal r.v.)
A = mean bar cross-section area (lognormal r.v.)
ε_{int} = mean initial strain in bar element (normal r.v.)
δ = coefficient of variation
σ = standard deviation

TABLE 3-2. LOADING RANDOM VARIABLES FOR 3D SPACE TRUSS.

Load	Type	mean (*kips*)	δ
S_1	Lognormal	10.0	0.25
S_2	Lognormal	10.0	0.25
S_3	Lognormal	500.0	0.25

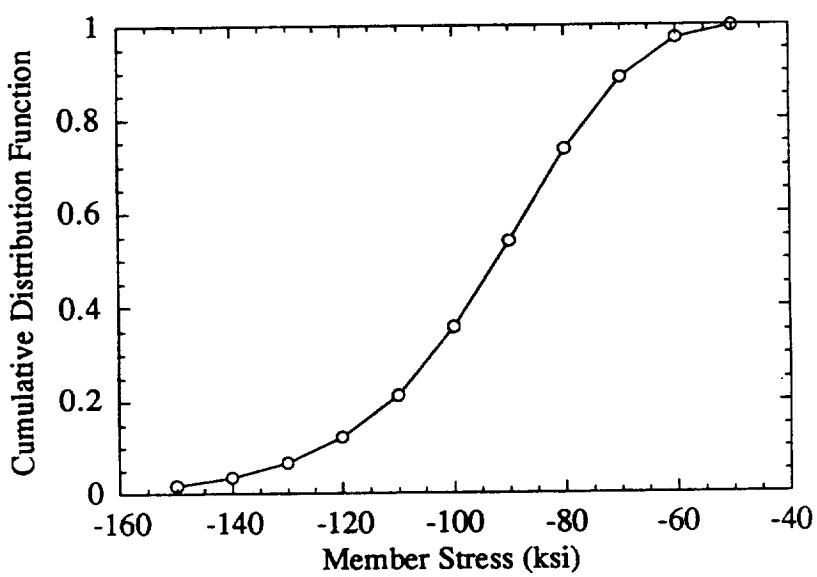


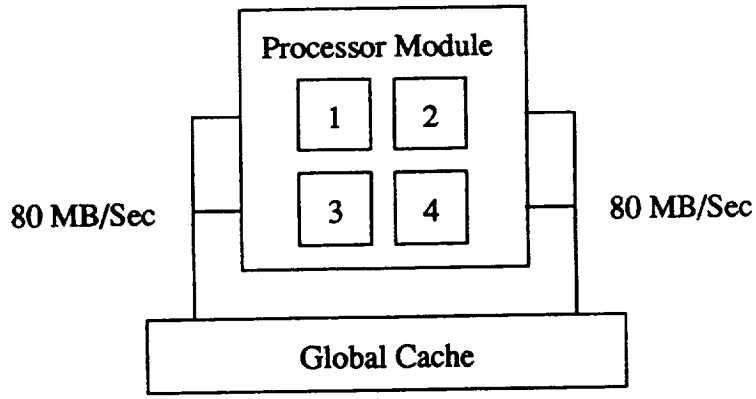
Figure 3-3. CDF for Stress in the Rear Vertical Element at the Base of the 3D Space Truss.

3.2.3.2 Parallel Performance

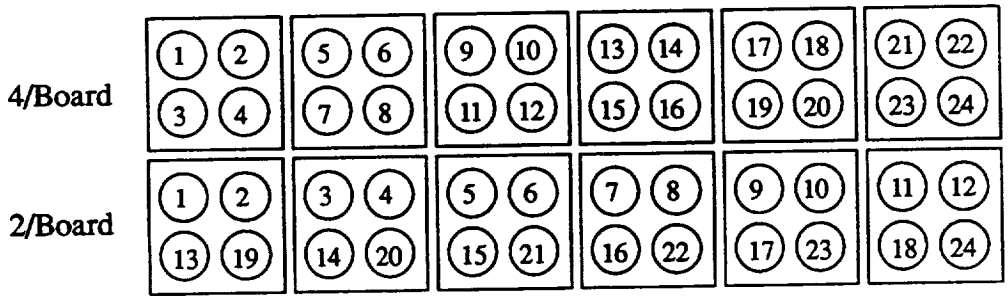
The memory requirement for this example to run on one processor is 312 *kB*. The size of the data-set, that needs to be replicated for each concurrent simulation trial execution is 56 *kB*. Thus, when using twenty-four processors there will be twenty-four simulation trials executing concurrently for a total memory demand of 1.6 *MB*. Since the memory size for the global cache (see Figure 3-1) is 4 *MB*, the code can be run on twenty-four processors without using the main-memory. Due to the high bandwidth from the processor module to the global cache, high efficiency and speedup can be obtained. Note that for a problem wherein adding processors causes a need to use slower memory, a drop in efficiency would be expected at the point that slower memory is required (we will see this effect in the next example problem).

The FX/2800 architecture supports four processors on each processor board, as illustrated in Figure 3-1. However, as detailed in Figure 3-4a, there are only two 80 MB/sec communication channels from each board to the global cache (through the crossbar switch). Thus, the manner in which processors are allocated will affect parallel efficiency. To investigate the influence of this effect and also to estimate the fraction of parallel overhead due to communication channel contention, two different configurations were used to allocate processors in the system — namely, two processors per board (2/board), and four processors per board (4/board). These configurations are illustrated in Figure 3-4b. In the 4/board configuration, four processors on a processor module are used first before allocating a processor on a new board. In the 2/board configuration, two processors on each module are allocated for the first twelve processors. After that, one more processor is allocated at each board. The 2/board configuration is designed to maximize bandwidth. If only two processors in a board are used, the bandwidth is 80 *MB/sec* for the board. On the other hand, if all four processors on a board are allocated, the bandwidth of the board reduces to 40 *MB/sec*. Hence, even in the 2/board configuration, we expect a degradation in parallel efficiency with more than twelve processors.

The comparison of the speedup factors for these two configurations along with the theoretical speedup is plotted in Figure 3-5. Theoretical speedup is computed using Equation 1-1. The difference in the value of speedup for the two processor allocation configurations is relatively small since the amount of data to be fetched directly from global cache is small. The closeness of the actual speedup to the theoretical speedup, even at twenty-four processors, demonstrates the high efficiency of the present approach for this small size problem. We should also point out that a significant portion of the loss of speedup at twenty-four processors is due to communication channel contention (*i.e.*, since we have four processors on each board sharing two channels). This is evident if we extend a straight line through the four-processor and eight-processor speedup values of the 4/board configuration out to twenty-four processors. The difference between the actual speedup and this straight line is approximately the true loss due to conventional parallel overhead (*i.e.*, management of concurrent processes, memory contention, and processor idling).



(a) Details of FX/2800 Processor Module.



(b) Processor Allocation.

Figure 3-4. Alliant FX/2800 Processor Allocation Configurations.

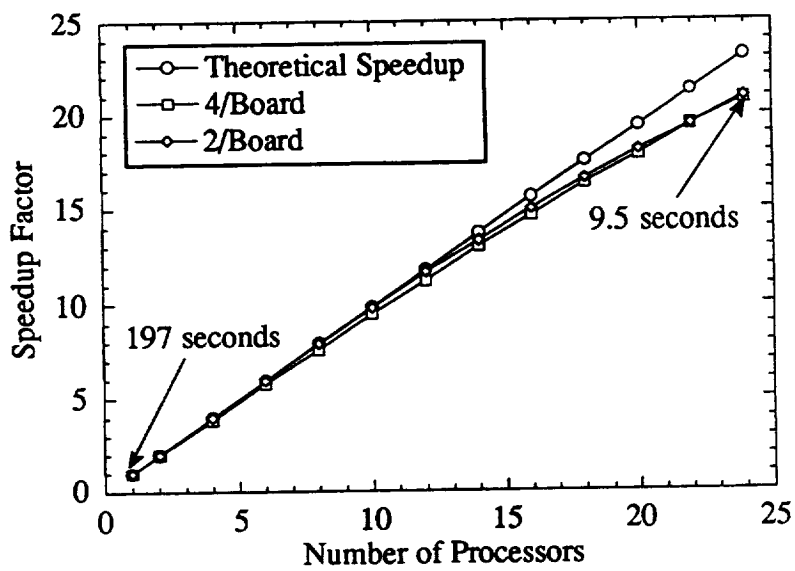


Figure 3-5. Parallel Speedup for 3D Space Truss on Alliant FX-2800.

Parallel efficiency (actual speedup divided by theoretical speedup, see Equation 1-2) for the two configurations is shown in Figure 3-6. This figure more clearly illustrates the difference in efficiency for the two configurations. The oscillation in efficiency for the 4/board configuration can be understood by examining Table 3-3. In Table 3-3 the value of efficiency for different numbers of processors is shown along with the corresponding value of the average bandwidth. Average bandwidth is a measure of the average data transfer rate. For example, when the code is run using six processors in the 4/board configuration (four processors on the first board and two processors on the second board), the average value of the bandwidth is given by (4 * 40 + 2 * 80)/6 = 53.33 *MB/Sec*. The oscillation in the value of efficiency shown in Table 3-3 is consistent with the variation in the value of the average bandwidth up to twelve processors. As the number of processors increases beyond twelve, conventional parallel overhead becomes dominant and efficiency declines monotonically.

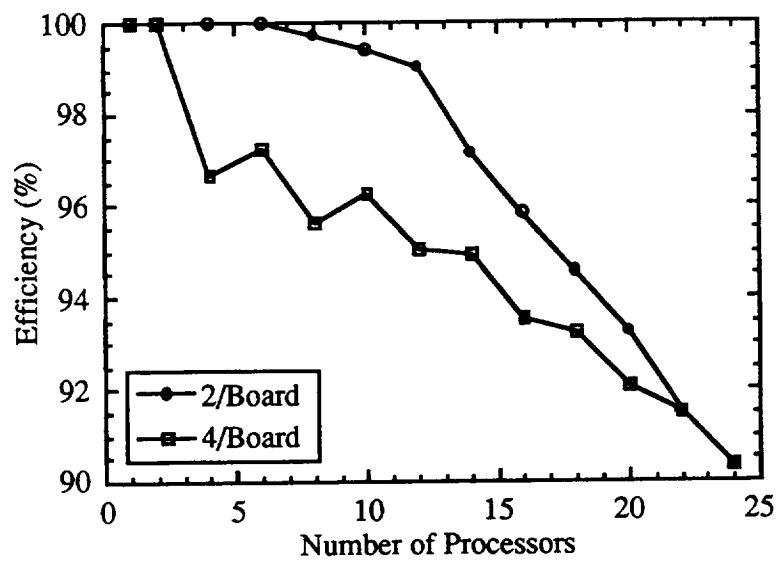


Figure 3-6. Parallel Efficiency for 3D Space Truss on Alliant FX/2000.

Referring again to Figure 3-6, due to the higher average bandwidth in the 2/board configuration than that in the 4/board configuration, the value of efficiency in the 2/board configuration is always higher than that in the 4/board configuration. Note the sharp drop in efficiency for the 2/board configuration when we go beyond twelve processors. This is the point where average bandwidth first reduces for this configuration. The steepness of the slope (beyond twelve processors) is in large part due to the memory channel contention. The gentler slope of the 4/board configuration is indicative of the efficiency loss due to conventional parallel overhead.

The previous timing studies used 10,000 Monte Carlo trials. Thus, we next investigated the affect of using a smaller sample size on the speedup. A comparison of the actual speedup for sample sizes of 1000 and 10,000 is shown in Figure 3-7. The loss in speedup is due to the fact

TABLE 3-3. EFFECT OF AVERAGE BANDWIDTH ON PARALLEL EFFICIENCY FOR 3D SPACE TRUSS ON ALLIANT FX/2800 (4/BOARD CONFIGURATION).

Number of Processors	Efficiency (%)	Average Bandwidth (MB/Sec)	Speedup
1	100.00	80	1.0
2	100.00	80	2.00
4	96.64	40	3.85
6	97.26	53.33	5.79
8	95.63	40	7.56
10	96.27	48	9.48
12	95.04	40	11.20
14	94.91	45.71	13.00
16	93.55	40	14.60
18	93.24	44.44	16.31
20	92.07	40	17.84
22	91.52	43.636	19.44
24	89.95	40	20.78

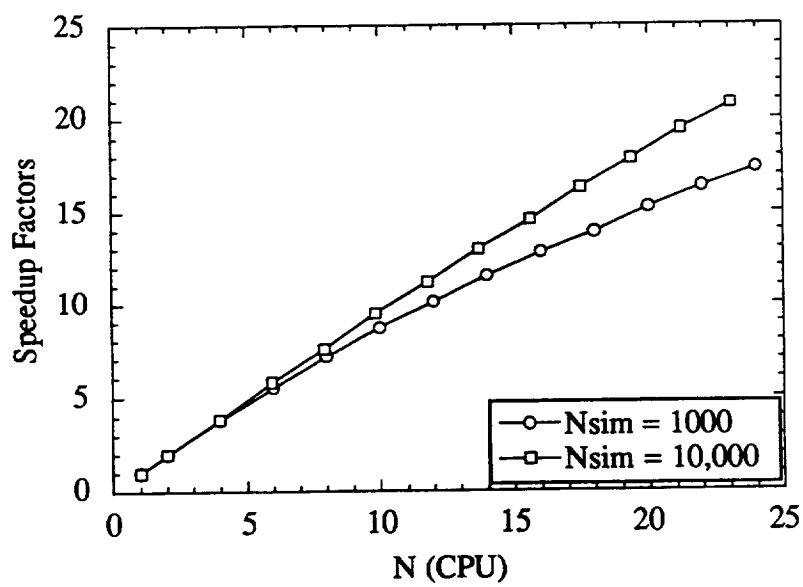


Figure 3-7. Variation of Parallel Speedup with Number of Simulation Trials.

that the fraction of sequential calculation (α in Equation 1-1) for 1000 samples is equal to 0.016529; whereas, the fraction of sequential calculations for 10,000 samples is 0.001698. Even though more than 98% of the work in the 1000-sample case (*i.e.*, $1 - \alpha > 0.98$) can be performed in parallel we observe a drop-off in the speedup for twenty-four processors. It is important to bear this realization of Amdahl's law in mind when designing a massively parallel implementation. Fortunately, for most problems of practical interest, α will be much less than the values here, as we will see in the next section.

3.2.4 Fatigue Crack Growth

3.2.4.1 Problem Description and Probabilistic Analysis Results

Fatigue failure is an important factor in the structural design and safety of many structures. Due to uncertainties in the cyclical mechanical and environmental loading, in the material properties of advanced materials (composites and ceramics) and in the shortcomings of analytical models, probabilistic methods are needed to ensure reliable designs and to assess the safety of existing structures. Probabilistic fatigue analysis is also particularly useful in combination with nondestructive evaluation techniques. Because the threshold of detection is substantially greater than flaw sizes that may lead to failure over the course of time, probabilistic description of flaws is necessary and inspection cycles should be set so that the reliability of an aging structure remains acceptable. Although a deterministic analysis can obtain an estimate of the fatigue life, the uncertainties in crack growth rates and the initial crack length detract from the usefulness of such solutions.

Due to the randomness in the location and orientation of the initial crack and the influence of the component boundary on the crack tip stress field, the resulting fatigue crack path is curvilinear. In order to characterize the curvilinear fatigue crack growth, a remeshing scheme together with crack tip singular elements has to be used in the finite element formulation. For problems of multiple fatigue cracks in which elastic interactions of a fatigue crack with micro-defects are treated, the remeshing scheme will be prohibitively complicated. In order to remedy this difficulty, the stochastic boundary element method (SBEM) developed by Lua, *et al.* [1992c] is employed to predict the fatigue life.

In the SBEM method, since the component boundary remains unchanged during the process of the fatigue crack growth, the influence matrices resulting from the component boundary are generated first and used in the subsequent fatigue crack growth stage. Further, when the material properties between the crack and the boundary are treated deterministically, these influence matrices are independent of the problem random variables. This treatment significantly reduces the computational effort of the probabilistic analysis since the matrices only need to be formed once, prior to execution of any probabilistic computations. This approach is a boundary element method analog to the finite element probabilistic substructuring method presented in Section 2.4. Notice here that all of the dominant uncertainties can still be treated (*i.e.*, loading, initial crack size and orientation, and the fatigue law parameters).

Figure 3-8 shows a square plate containing a single edge crack with a random initial crack length and crack angle subjected to a uniformly distributed load. The plate geometry, initial crack location, final crack size, and material constants are deterministic parameters given by

$$L = W = 2.5\ in,\ x_0 = 1.25\ \text{in},\ y_0 = 0.0\ in \tag{3-1}$$

$$a_f = 0.5\ in,\ \mu = 80000.0\ psi,\ \nu = 0.3 \tag{3-2}$$

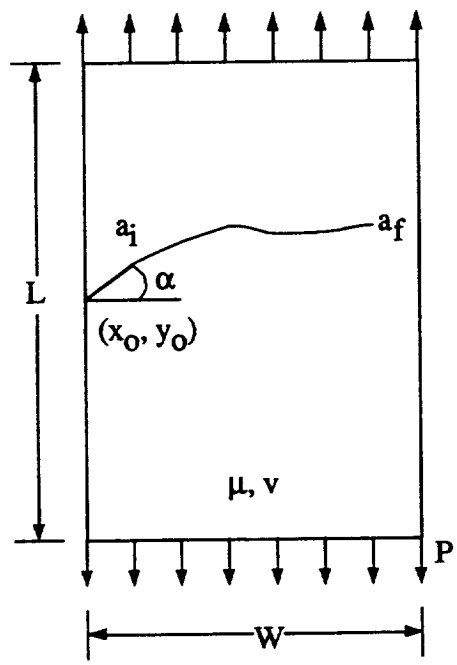


Figure 3-8. Single-Edged Fatigue Crack Growth Example.

where μ is the shear modulus and v is the Poisson's ratio. The crack geometry (a_i, α) external load (P), and fatigue parameters (D, n) are assumed independent random variables with specified probability density functions. The statistical parameters of the input random variables (mean and standard deviation) along with corresponding distribution functions are listed in Table 3-4. As shown in Table 3-4, the initial crack size, a_i, has the largest dispersion (COV ≈ 60%).

Figure 3-9 shows the probability density function of the initial crack size. The detection threshold, which is equal to 7.5E-03, represents the lower limit of the device to detect the presence of a small initial crack. Below the detection threshold the probability density is assumed uniform; above the threshold the probability density decays linearly to zero, representing false negatives of the inspection technique.

TABLE 3-4. FATIGUE LIFE RELIABILITY PROBLEM RANDOM VARIABLES.

Random Parameters	Distribution	Mean	Standard Deviation σ
Crack Angle, α (*radians*)	Uniform	0.0	0.4534
Initial Crack Length, a_i, (*inches*)	Uniform with Tail	5.833E-03	3.584E-03
Fatigue Parameter, D	Lognormal	3.77E-07	1.885E-08
Fatigue Parameter, n	Lognormal	3.60	0.18
Applied Stress, P (*ksi*)	Lognormal	11.0	1.1

The performance function for the fatigue problem is

$$g(\mathbf{b}) = g(T(\mathbf{b})) = T(\mathbf{b}) - T_s \tag{3-3}$$

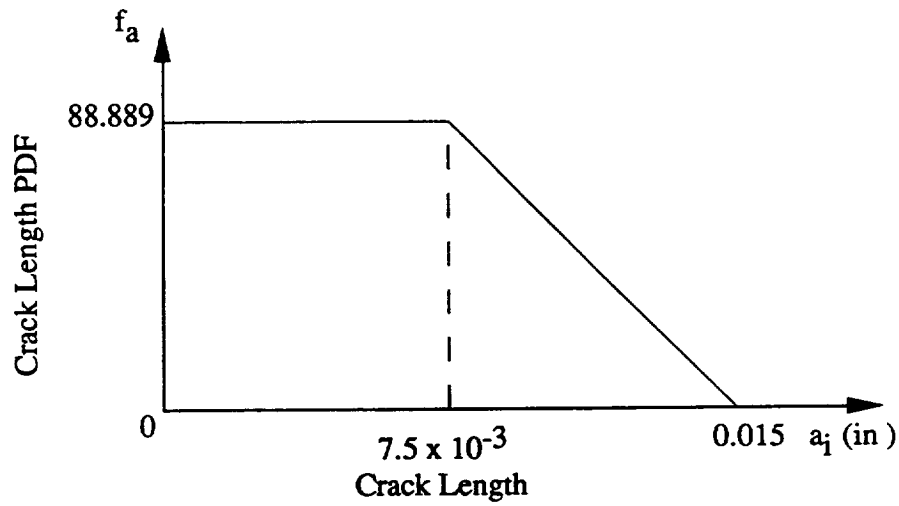


Figure 3-9. PDF of Initial Crack Length.

where **b** is the random vector, T_s is the required service life, and T is the predicted fatigue life, which is estimated using the Paris-Erdogan law [Paris and Erdogan, 1963]:

$$T = \int_{a_i}^{a_f} \frac{da}{D(\Delta K_{eq})^n} \tag{3-4}$$

where D and n are fatigue parameters, ΔK_{eq} is the range of equivalent Mode I stress intensity factor in a cycle given by

$$\Delta K_{eq} = K_{eq}^{\max} - K_{eq}^{\min} \tag{3-5}$$

The quantities $K_{eq}^{\max}$, $K_{eq}^{\min}$ in Equation 3-5 are the minimum and maximum equivalent Mode I stress intensity factors associated with the minimum and maximum cyclic applied stresses, respectively. If $K_{eq}^{\min} = 0$, Equation 3-5 reduces to

$$\Delta K_{eq} = K_{eq}^{\max} \tag{3-6}$$

The relation between the equivalent Mode I SIF (K_{eq}) and SIFs (K_I and K_{II}) is represented by

$$K_{eq} = \cos^3 \frac{\alpha}{2} K_I - 3\cos^2 \frac{\alpha}{2} \sin \frac{\alpha}{2} K_{II} \tag{3-7}$$

where α is the crack growth direction determined by the crack direction law [Erdogan and Sih, 1963]

$$K_{\mathrm{I}} \sin \alpha + K_{\mathrm{II}} (3 \cos \alpha - 1) = 0 \tag{3-8}$$

The determination of the history of K_{eq} with the fatigue crack length (a) is the key task to predict the fatigue life T (Equation 3-4). Since the fatigue life depends on the crack path, the problem has to be solved in steps. For a given initial crack length a_i and final crack length a_f, nineteen steps are used to grow the crack from a_i to a_f. The initial crack length a_i is discretized into ten boundary elements and five elements are used in the subsequent crack growth stages. The outer boundary of the plate (see Figure 3-8) is discretized into sixty-nine elements.

MCS was used to perform the probabilistic analysis, and the statistics of the fatigue life are given in Table 3-5. The relatively large dispersion in the fatigue life (COV = 51%) demonstrates the importance of using the stochastic approach in addressing fatigue problems. The complexity and variability of a composite material will introduce additional uncertainty to the problem. The cumulative distribution function of the fatigue life T is plotted in Figure 3-10 for both 200 and 1000 simulation histories. As expected, 200 histories gives accurate results for the probability levels plotted here (approximately 0.05 to 0.95 for Nsim = 200). For the parallel performance studies to follow we will, therefore, use 200 simulation histories. Samples of the random curvilinear fatigue crack paths obtained on different Monte Carlo trials are plotted in Figure 3-11. Due to the external Mode I loading, the crack path becomes rectilinear after a few steps of the crack growth.

TABLE 3-5. STATISTICAL PARAMETERS OF THE FATIGUE LIFE.

Mean (T)	Standard Deviation	COV
1.32E06	0.676E06	51%

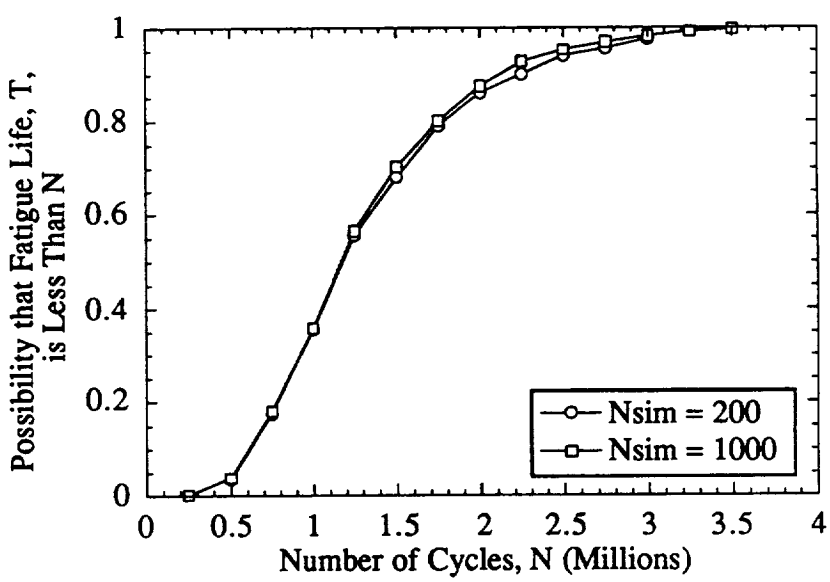


Figure 3-10. Fatigue Life CDF.

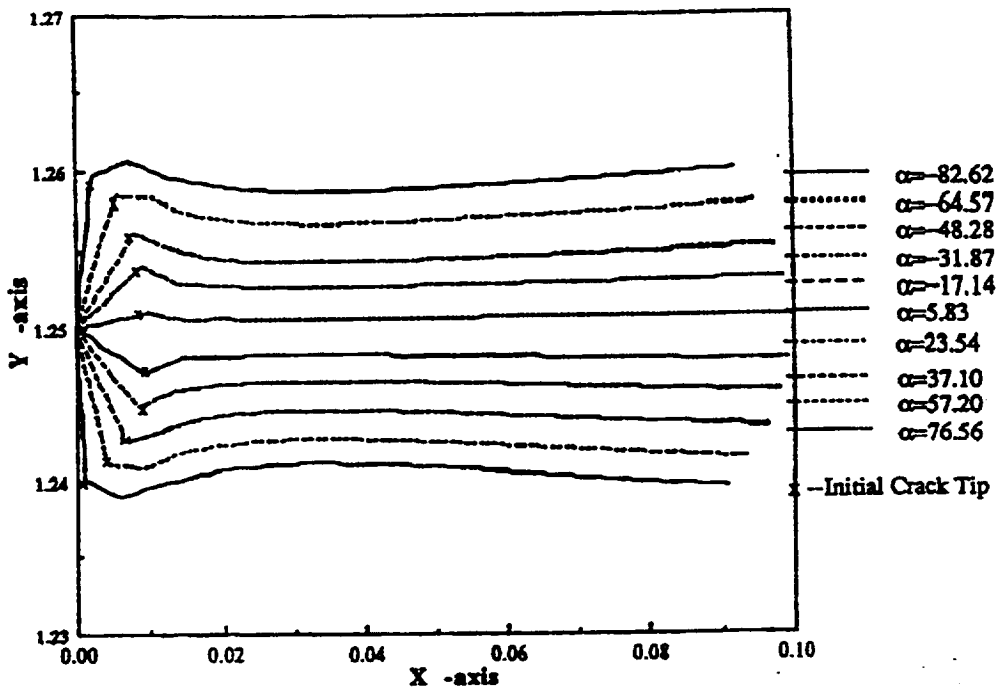


Figure 3-11. Fatigue Crack Paths from Random Initial Orientation.

From sensitivity analysis of the fatigue life with respect to the problem random variables, we can conclude that the fatigue parameter, D, and the initial crack length, a_i, are the dominant random parameters in the fatigue crack reliability analysis. The response sensitivity to the initial crack angle, α, is extremely small over the entire range of failure probability. The lack of importance of α is mainly due to the dominance of the present Mode I loading.

3.2.4.2 Parallel Performance

The present fatigue problem is much more memory intensive than the 3D truss problem. The memory requirement for this problem to run on one processor is 4 *MB* and the size of the data-set, which needs to be replicated for each concurrent simulation execution is 3.3 *MB*. When using twenty-four processors there will be twenty-four simulation trials executing concurrently for a total memory demand of 80 *MB*. Since the memory size for the global cache (see Figure 3-1) is 4 *MB* using more than one processor requires that main memory be used. Thus we would expect to see an immediate drop in efficiency when going to two processors.

Similar to the 3D truss problem, two configurations are used to allocate work load to the processors; 2/board and 4/board (see Figure 3-4). The comparison of efficiencies for both configurations is shown in Figure 3-12.

For the 2/board configuration we observe an initial drop in efficiency going from one to two processors. As mentioned above, using two processors requires the use of main memory;

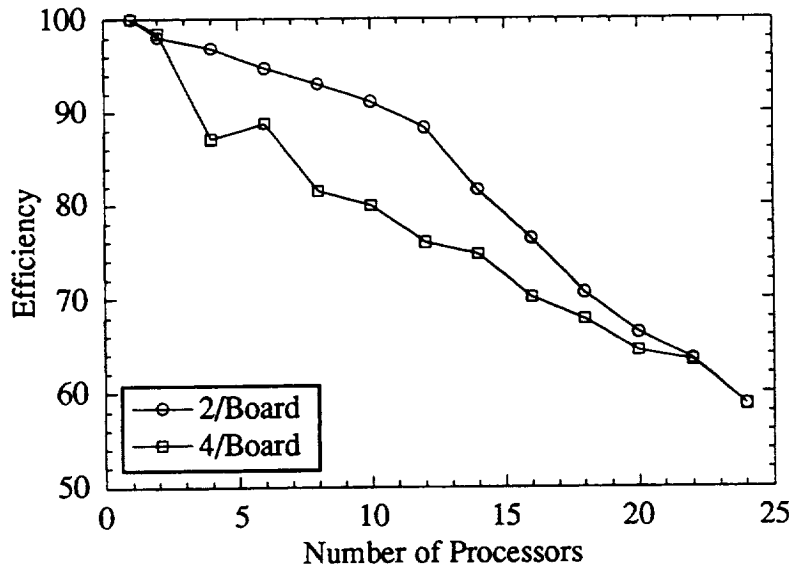


Figure 3-12. Parallel Efficiency for Fatigue Reliability Problem on Alliant FX/2800.

whereas with one processor the entire problem fits in the cache. When using from two through twelve processors we see a gradual decrease in efficiency due to parallel overhead (*i.e.*, management of concurrent processes, memory contention, and processor idling). Beyond twelve processors we see a steeper downward slope that is mainly due to competition of three or four processors on a single board for two communication channels (see Section 3.2.3.2). Conventional parallel overhead (*i.e.*, management of concurrent processes, memory contention, and processor idling) also contributes to the loss in efficiency.

The 4/board configuration efficiency results are also shown in Figure 3-12. As expected, the 4/board efficiency is always below the 2/board configuration's efficiency. The oscillation in the value of efficiency is consistent with the oscillation in the average bandwidth, as was described in detail for the 3D truss problem (see Section 3.2.3.2), for up to twelve processors. Beyond twelve processors parallel overhead becomes dominant and efficiency declines monotonically. The gentler downward slope of this curve, as compared with the 2/board curve, is more indicative of efficiency loss from conventional parallel overhead (since the slope of the 2/board curve is dominated by communications channel contention).

The speedup factors for these two configurations along with the theoretical speed-up are shown in Figure 3-13. The value of α (i.e., the fraction of the work load that cannot be done in parallel, Equation 1-1) is extremely small (α = 0.000044) due the large cpu time required for each simulation trial. The resulting theoretical speedup (see Equation 1-1) is almost linear as shown. The value of the speedup for the 2/board configuration is always higher than the value for the 4/board configuration due to the larger value in the average bandwidth. As the number of processors increases, the speedup curves for both configurations approach a horizontal line. This implies that increasing the number of processors will not result in additional speedup.

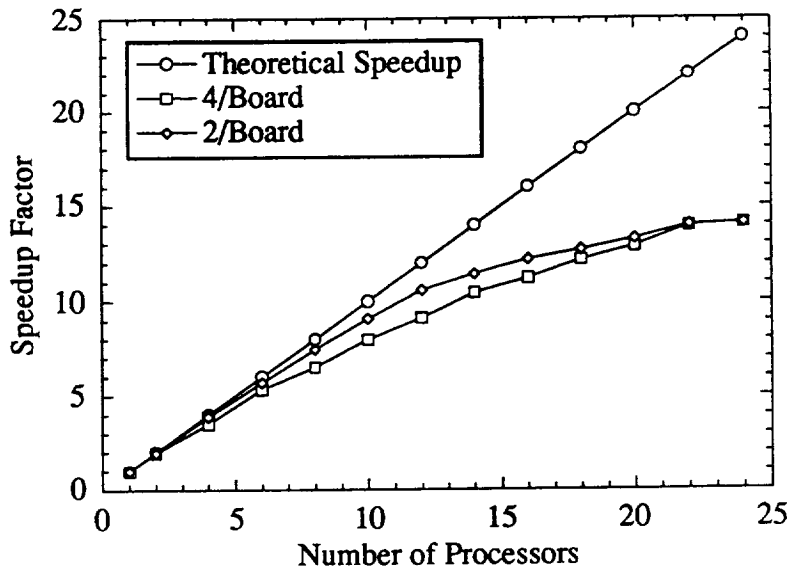


Figure 3-13. Parallel Speedup for Fatigue Reliability Problem on Alliant FX/2800.

We should also point out that a portion of the loss of speedup at twenty-four processors is due to communication channel contention (since we have four processors on each board sharing two channels). This is evident if we extend a straight line through the four-processor and eight-processor speedup values of the 4/board configuration out to twenty-four processors. The difference between the actual speedup and this straight line is an approximate measure of the true loss due to conventional parallel overhead.

It is important to recognize that for larger problems memory requirements could exceed the size of physical memory as the full complement of processors is utilized. Thus, secondary storage would have to be used (disk paging on the Alliant) resulting in a significant increase in the apparent overhead and severely limiting the concurrency speedup. In this case, multiple levels of parallelism could be exploited to reduce the problem memory requirements.

3.3 DISTRIBUTED-MEMORY ARCHITECTURE

3.3.1 Message-Passing Programming Paradigm with CS Tools

Communication between processors is the key issue in parallel implementation on a distributed-memory machine, as opposed to competition for communicating with and the use of global memory on the shared-memory machine. The conventional approach to developing code on a distributed-memory multiprocessor uses a message-passing paradigm. Message passing is accomplished using a set of library functions that are generally hardware-specific (some software packages do exist, however, that will port to more than one machine, *e.g.*, APPL and EXPRESS); however, the principles of implementation are generic. Here our purpose is to investigate how to parallelize a probabilistic analysis code on a distributed-memory machine using the message-

passing paradigm. For this investigation we have selected the CS Tools library of functions that operate on the Meiko Computing Surface hardware.

It is useful to view the distributed system as a computing surface network (CSN). CSN communication between processors occurs through transports. Each transport in the network has a unique address, which must be used by the sender of a message to identify the target of the communication. As shown in Figure 3-14, both Processor 1 and Processor 2 create a connection between the processor and the CSN.

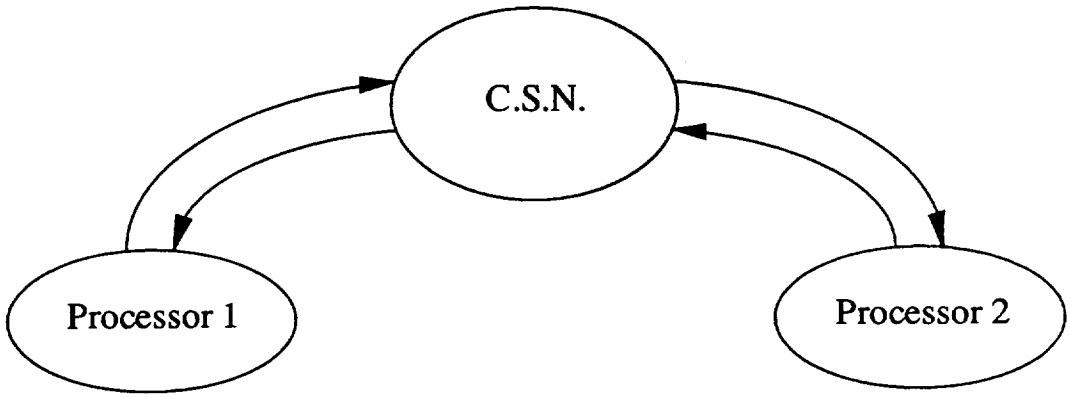


Figure 3-14. Distributed-Memory Software Development: Message-Passing Paradigm with CS Tools.

The communication between processors is analogous to making a telephone call. Using this analogy, a person represents a processor, a telephone line represents a transport, the telephone number is the Net Id of a transport and the telephone exchange represents the CSN. The CS Tools Communications Library provides routines to create a connection between the processor and the CSN (csnopen()); to register the transport's Net Id (csnregname()); and to find the Net Id of another processor (csnlookupname()). Having established the Net Id of the receiver's transport, the sender can pass its data by using either the blocking transmission routine csntx() or the non-blocking transmission routine csntxnb(). Similarly, the receiver can receive the data by either calling the function, csnrx(blocking), or csnrxnb(non-blocking). The main difference between blocking and non-blocking is that the blocking communication call will always suspend the calling process until the transfer has completed, whereas a non-blocking communication call will return almost immediately. In order to minimize processor idle time, non-blocking data transmissions are employed in this study.

In the parallel computation, the total amount of computational work has to be distributed and allocated to the available processors. This is achieved by using the master-slave model as shown in Figure 3-15. The function of Master.F is to generate tasks and to perform the load balancing and distribution of jobs to separate processes. Each slave process assigned to a given processor performs its task (*e.g.*, simulation trial in MCS or sensitivity calculation in FORM/SORM), and sends the result back to the master processor for post-processing. As

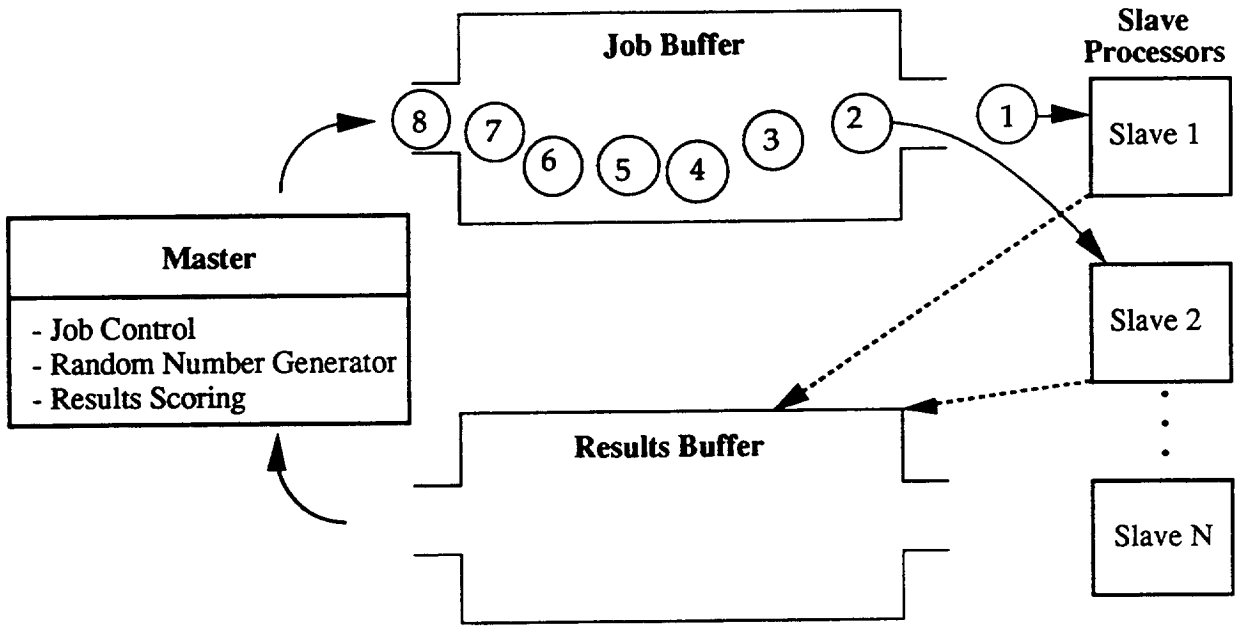


Figure 3-15. Master-Slave Model for Distributed-Memory System.

non-blocking communications always return control to the caller without waiting for the transmission to complete, the non-blocking communication is used for the data passing and receiving between processors to enhance efficiency. As shown in Figure 3-15, the master continuously generates the jobs and puts them in its job buffer for a slave to pick up and consume. Similarly, a slave processor keeps sending results to the results buffer for the master to pick up. Since both csnrxnb and csntxnb only queue buffers for reception or transmission of messages and return control to the caller immediately, an additional function, csntest, must be used periodically to determine the completion status.

A flowchart of the code segment for non-blocking communication between the master and a slave is shown in Figure 3-16. As shown in the figure, the master puts N jobs in the job buffer using non-blocking transmission csnrxnb(), where IS_M and ID_S(j) are the ID number of the master and the jth slave on the CSN, respectively. After that, the master queues a result buffer. The completeness of both the non-blocking transmission and reception are checked by calling the routine, csntest. Based on the sign of the tag returned by csntest (itag), it can be determined whether the sending or receiving has been completed. For the present Monte Carlo Simulation implementation, the subroutine nextjob involves the random number generation and the subroutine nextresult performs the scoring and summation.

Each slave process queues a job buffer by using csnrxnb(). The completeness for receiving and sending is then checked using csntest(). For the case when the receiving is completed (itag = 1) the slave will consume the job and send the results back to the master. The subroutine slave work performs all the deterministic analysis. For the case where the transmission of results is completed (itag = -1), the result buffer is re-queued to store the next set of results.

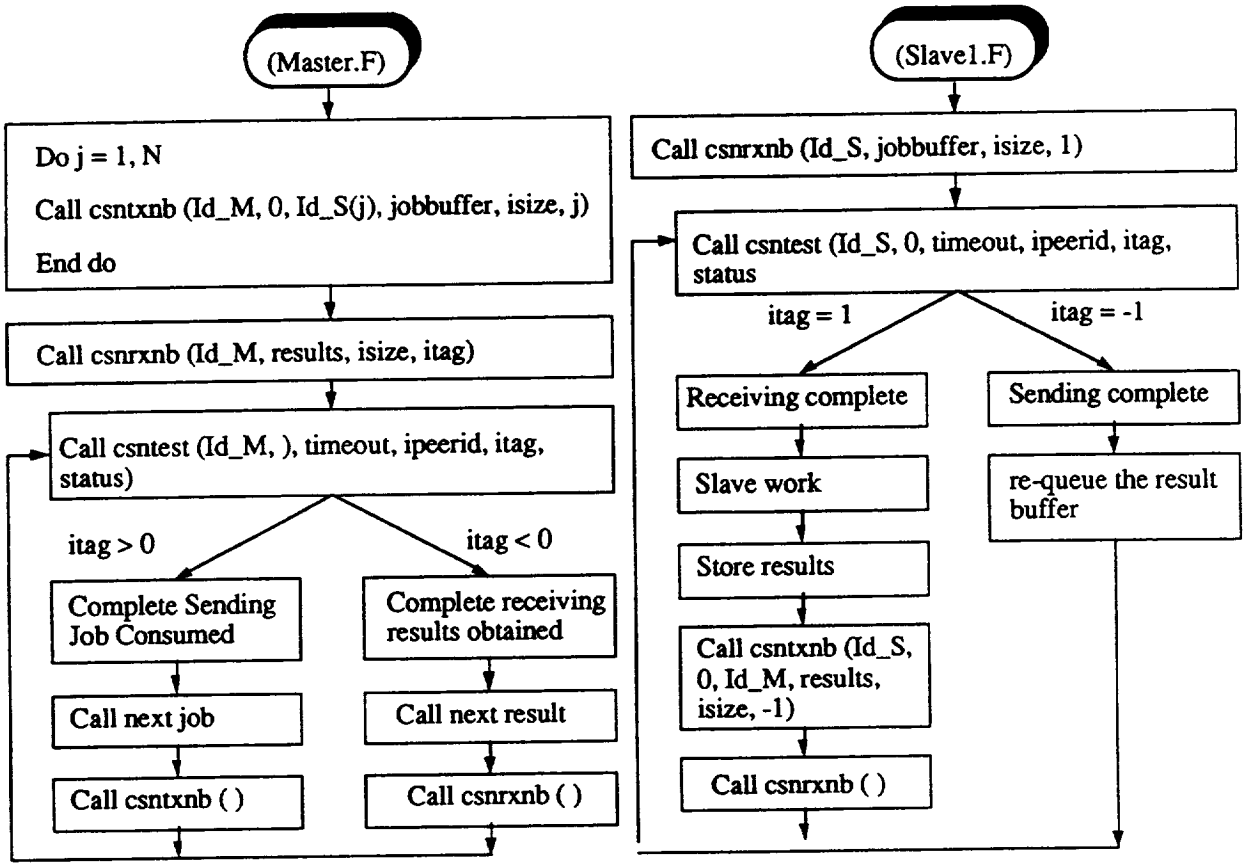


Figure 3-16. Implementation of Master-Slave Model for Parallel Fatigue Reliability Analysis with CS Tools.

In the code, the non-blocking transmission is organized in such way that it does not matter if the solution time of the jobs has a large variation. In other words, one processor can process many small jobs without being blocked by another processor taking a long time to process a single job; thus a natural load balancing occurs.

The specific application of the Master-Slave model to Monte Carlo simulation can be described as follows. First, the master code must generate the random parameters. This is performed in the routine nextjob. The generated job is processed by any slave that happens to be free at the time. When the number of simulations reaches the total number of desired histories, the master code sends a synchronization signal to each slave processor. Once the master processor receives a response to the synchronization signal from each slave processor, the process of scoring and statistical analysis on the simulation data is performed. Finally, the master transport is closed by calling the library routine, csnclose.

The slave processor first receives the deterministic problem parameters sent by the master by calling the routine slaveinit (not shown in Figure 3-16). Next, the slave processor checks its own job buffer to see whether there is a real job or an end signal (*i.e.*, the synchronization signal) sent by the master. For a real job, the slave processor first finds a free result buffer and gets the

set of random parameter realizations from its job buffer. Then, the slave processor performs the deterministic analysis by calling the slavework routine and sends back the result to the master. Once the result has been sent, both the job buffer and result buffer are re-queued for the next simulation. If the slave processor receives the end signal, it sends the same signal back to the master to allow for the final scoring and statistical analysis.

3.3.2 Virtual Shared-Memory Programming Paradigm with C-Linda

3.3.2.1 Overview

As an alternative to the message passing paradigm we next investigated the virtual shared-memory paradigm. The virtual shared-memory approach generally carries additional overhead but it has the advantage of portability across a wide range of parallel platforms and memory architectures. Given that software development and maintenance is an expensive and time-consuming process, portability is one of the key issues in software design and implementation for parallel architectures today. That is, it is desirable for software systems to be able to run on any reasonable parallel computer with little or no modification. This will be critical to enhancing the commercialization potential of this SBIR. Another important factor is that the parallel language be general enough to support the programming model appropriate for the problem at hand. The virtual shared-memory approach will readily support the multi-level parallelism that will be required to achieve large-scale parallelization.

For implementation of the virtual shared-memory paradigm on a distributed-memory machine we selected the C-Linda language. C-LINDA is a toolkit of routines that implements the virtual shared-memory model, known in C-Linda as "tuple space." These routines replace the low level message passing and synchronization mechanisms found in other parallel programming methods. C-LINDA consists of a few simple operations which control process creation and coordination and are orthogonal to the base language in which it is embedded.

C-LINDA was developed specifically to solve the problem of portability among different parallel computer architectures. C-LINDA has implementations on both shared-memory machines and distributed-memory machines. Another feature of C-LINDA is that implementations exist for local-area networks. Even though the communications over a local-area network is substantially slower than a parallel machine, coarse grain problems perform satisfactorily on local-area networks because the communication to computation ratio is very small. Many sites have computation intensive problems and, while lacking parallel computers, have networks of occasionally under-used or idle workstations. Converting unused workstation resources into performance gains is an economical and attractive possibility.

To discuss the implementation of any parallel processing application using LINDA, it is useful to understand the abstract programming models used in designing parallel software. In terms of software design for parallel computers, we can envision parallelism in terms of three conceptual classes: a program's results, a program's agenda of activities, or an ensemble of specialists that collectively constitute the program [Carriero and Gelernter, 1989]. In result

parallelism the application is designed around the data structure yielded as the ultimate result. Result parallelism focuses on the shape of the final product and we get parallelism by computing all elements of the result simultaneously. With agenda parallelism the application is designed around a list of activities and parallelism is achieved by assigning many workers to each task. The third class of parallelism, the ensemble of specialists, results when the application is a model of a logical network of some type, with each node performing a special task, parallelism results from each node performing a special and distinct function simultaneously.

Corresponding to these three classes of parallelism are three programming methods or techniques for translating concepts into working programs: (1) message passing, (2) live data structures, and (3) distributed data structures. In message passing we create many processes and enclose every data structure in some process. Figure 3-17 illustrates message passing where processes are round, data objects are square and messages are oval. With this method no data objects are shared and processes communicate by passing messages to each other. This message passing is explicitly specified in the program code. Message passing is often used to implement the ensemble of specialists class of parallel problems. The live data structure technique is at the other end of the spectrum from message passing and is used to implement result parallelism. A live data structure program, illustrated in Figure 3-18, is built in the shape of the resulting data structure. Each process is an independent entity and when the process is complete it returns one part of the result. Live data structure processes do not pass messages, rather they simply refer to each other as elements of some data structure. The message passing and live data structure techniques are similar because all of the data is distributed among the processes and there is no global memory. In message passing, however, process creation and communication is handled explicitly by the programmer. In live data structure programs, processes are created implicitly in the course of building the data structure and communicate with other implicitly by referring to elements of the data structure. In between allowing all the data to be absorbed into the process structure or all processes resulting in data elements is an intermediate technique which maintains the distinction between a group of data objects and a group of processes. These distributed data structure programs share data and communicate by placing data objects in a common shared area as shown in Figure 3-19. Agenda parallelism maps naturally onto distributed data structure programs. In this type of parallelism many workers work on what is in effect a single job and any worker will be willing to pickup any subtask.

As can be seen from the previous discussion, parallel computational algorithms usually fall naturally into one of the three conceptual classes: (1) result parallelism, (2) agenda parallelism, or (3) specialist parallelism. The programming technique then follows from the conceptual class. Linda's virtual shared-memory ("tuple space") model allows for programs to be easily developed with either message passing, live data structures, or distributed data structures.

Figure 3-20 illustrates the essentials of the virtual shared-memory approach. Tasks and data, called tuples, are place into the tuple space (*i.e.*, the virtual shared-memory) to be worked on by available processors. There are two types of tuples: process tuples that generate use, and consume tuples; and data tuples that are essentially passive data to be operated on. Since process

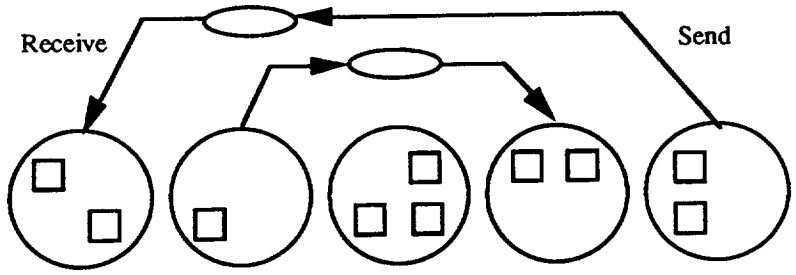


Figure 3-17. Message Passing [Carriero and Gelernter, 1989].

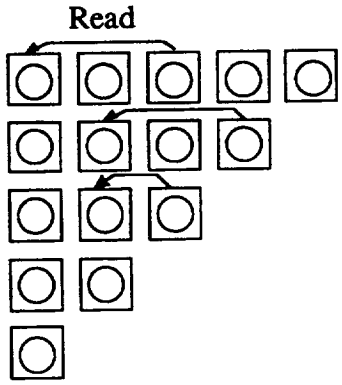


Figure 3-18. Live Data Structures [Carriero and Gelernter, 1989].

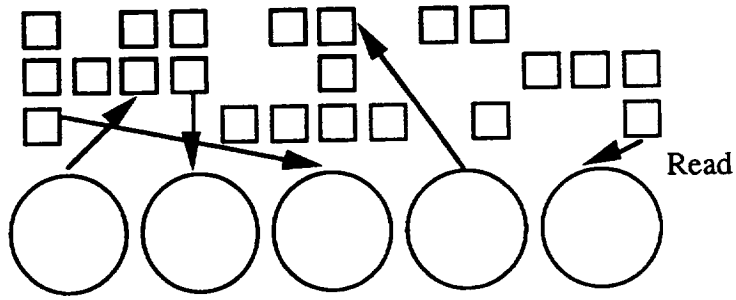


Figure 3-19. Distributed Data Structures [Carriero and Gelernter, 1989].

tuples are worked on by the next available processor a natural load-balancing is achieved. Table 3-6 shows the four basic C-Linda functions used to manipulate tuples.

3.2.2.2 Application of C-Linda for Multi-Level Parallelism

For the fatigue reliability problem, parallelism can be identified at several levels. For this Phase I implementation we used C-Linda to exploit two levels of parallelism: the Monte Carlo simulation loop and the Green's function computations to obtain the influence coefficients for displacement along the plate boundaries due to crack opening displacement (these influence coefficients must be recomputed for every Monte Carlo trial). The computations are performed in parallel in C-Linda using agenda type parallelism which is analogous to the Master-Slave model used in the message passing paradigm. We will describe first how the Monte Carlo loop is parallelized using C-Linda. This will be followed by a description of how the influence coefficient computations are parallelized.

In the original sequential MCS code, the subroutine determ (for deterministic evaluation of the fatigue life for a set of input parameters) is called once for each Monte Carlo history:

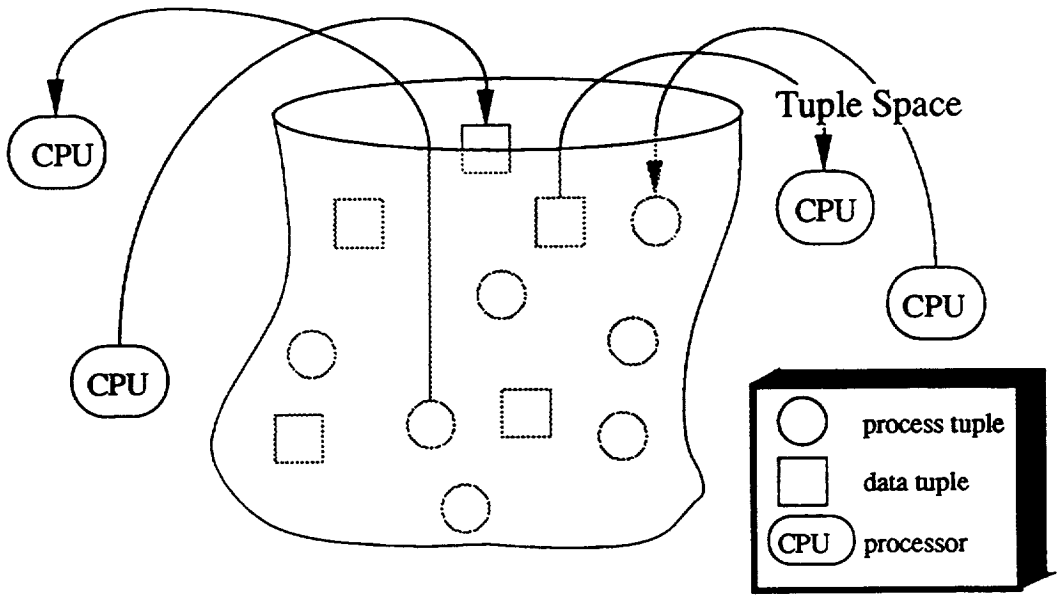


Figure 3-20. Distributed Memory Software Development: Virtual Shared-Memory Paradigm with C-Linda.

TABLE 3-6. BASIC LINDA FUNCTIONS.

Function	Description
out (t)	Evaluate t and place it in tuple space.
in (t)	Withdraw t from tuple space.
rd (t)	Read t from tuple space.
eval (t)	Evaluate t after it is placed in tuple space.

```
Original Sequential FORTRAN ...

   do 140 npce=1,nr

     call determ(npce,xci(npce),yci(npce),cth(npce),cai(npce),
  &caf(npce),dfp(npce),fpn(npce),rff(npce),ncdf,tsv,fail(1,npce),
  &st,sst,x,y,xm,ym,cx,cy,ssol,dsol,g,h,kode,fil,
  &2*nbp)

140   continue
```

In the LINDA Master-Worker implementation, the Master process places the initial history number into Linda's virtual shared-memory — tuple space — and starts a Worker process on each node.

In the LINDA MASTER ...

```
/*-----------------------------------------------
C
C  Output initial history number
C
   ------------------------------------------------*/
out ("histnum", 1);
/*-----------------------------------------------
C
C  START PROCESS ON EACH WORKER
C
   ------------------------------------------------*/
for ( i=0; i<numworkers; i++ )
{
   eval("worker", worker());
}
/*-----------------------------------------------
C
C  GET HISTORY DATA FROM WORKERS
C
   ------------------------------------------------*/
for (i = 1; i <= nr; ) /* for i= num of hists */
{
   stat=inp("hist", i, ? dtemp1, ? dtemp2, ? xcm, ? ycm, ? xc, ? yc,
             ?  cgdir, ? sif, ? cod, ? ftl, ? fail[i-1] );
   if ( stat ) /* if worker history data available */
   {
      st += dtemp1;
      sst += dtemp2;
      output(xcm, ycm, cod, sif, cgdir, xc, yc, & ftl, & idiv,
             & nstp, &ncp, & i);
      i++;
   }
   else /* else let Master calculate one history */
   {
      if ( moredata==1 )
      {
         in("histnum", ? itemp);
         out("histnum", itemp + 1);
         if (itemp <= nr )
         {
            mworker(itemp);
         }
            else
         {
            moredata=0;
         }
      }
   }
}
```

Each Worker process simply retrieves the next history number to be calculated and calls the same determ subroutine called in the original sequential code. When determ returns, the Worker writes the history data to the tuple space and attempts to work on another history.

```
In the LINDA WORKER ...
while ( 1 )
{
   in ("histnum", ?histnum);
   out ("histnum", histnum+1);
   if ( histnum > nr )
   {
      break;
   }
   determ(&histnum, &xci[histnum], &yci[histnum], &cth[histnum],
          &cai[histnum], &caf[histnum], &dfp[histnum], &fpn[histnum],
          &rff[histnum], &ncdf, &tsv, &fail[histnum][0], &st, &sst,
          x, y, xm, ym, cx, cy, ssol, dsol, g, h, kode, fil, &itemp,
          xcm, ycm, xc, yc, cgdir, sif, cod, &ftl, &n, &l, nc, &m,
          &ge, &xnu, &nstp, &ncp, &idiv);
   out("hist",histnum, st, sst, xcm, ycm, xc, yc, cgdir, sif, cod,
        ftl);
}
```

As the Workers place data from each history into the tuple space, the Master retrieves and processes (*i.e.*, scoring) the data. If at any time there is no history data for the Master to process, the Master will itself pitch in and perform a history calculation in order to make full use of computing resources at all times. The entire process continues in parallel until all histories have been calculated.

The above programming model is directly applicable for other probabilistic analysis methods (*e.g.*, FORM/SORM/FPI, response surface, etc.) with only minor modification, since they all require repeated independent evaluations of the problem performance function.

In the event that the ratio of the number of available processors to the number of independent probabilistic analysis solutions is large (*e.g.*, for small sample MCS or non-MCS methods), or the problem has large memory requirements it would be advantageous to exploit additional levels of parallelism. We illustrate below how the virtual shared-memory model is well-suited to invoking multiple levels of parallelism.

In the Monte-Carlo loop parallelization described above, the Linda eval function was used to create process tuples (workers) to perform the independent history calculations. The following code is a section of the determ routine (for deterministic evaluation of the fatigue life for a set of input parameters) that is called once from each "evaled" worker to perform history calculations. The objective here is to now add another level of parallelism and perform one of the sub-loop calculations in parallel.

We illustrate here how we can parallelize the "do 40 k =" and "do 40 j =" nested loop (the "boxed" code shown below). In this nested loop, the subroutine udgrma (evaluation of boundary node influence coefficients for displacement along the plate boundaries due to the

crack opening displacement) is called repeatedly to perform n*ncp0 independent calculations. In the following code, the p_udgrma routine (parallel udgrma) replaces the boxed code to perform the nested loop in parallel.

```
          do 20 i=1,nstp
          nce=((i-1)*ncp+idiv)*2
          nsize=2*n+nce
          do 9 k=1,nsize*(nsize+1)
9         sto(k)=0.d0
          if(i.gt.2) then
          ne(i)=ne(i-1)+ncp
          end if
          if(i.eq.1) then
          dcl0=dil
          ncp0=idiv
          else
          dcl0=dcl
          ncp0=ncp
          end if
          do 25 j=1,ncp0
          xc(ne(i)+j)=xc(ne(i)+j-1)+dcl0*dcos(cgdir(i))
          yc(ne(i)+j)=yc(ne(i)+j-1)+dcl0*dsin(cgdir(i))
25        continue
          do 26 j=ne(i),ne(i)+ncp0-1
          xcm(j)=(xc(j)+xc(j+1))/2
          ycm(j)=(yc(j)+yc(j+1))/2
26        continue
          call p_udgrma(xc, yc, udg, ne(i), ncp0, n)
          do 40 k=1,n
          do 40 j=ne(i),ne(i)+ncp0-1
          call udgrma(xm(k),ym(k),xc(j),yc(j),xc(j+1),yc(j+1),
     &    udg((2*k-1),(2*j-1)),udg((2*k-1),2*j),
     &    udg(2*k,(2*j-1)),udg(2*k,2*j),ge,xnu)
40        continue
          do 45 k=1,ne(i)+ncp0-1
          call angel(xcm(k),ycm(k),xc(k+1),yc(k+1),alph)
          do 45 j=ne(i),ne(i)+ncp0-1
          call sdtrma(xcm(k),ycm(k),alph,xc(j),yc(j),xc(j+1),yc(j+1),
     &    sdt((2*k-1),(2*j-1)),
     &    sdt((2*k-1),2*j),sdt(2*k,(2*j-1)),sdt(2*k,2*j),ge,xnu)
45        continue
          do 60 lc=1,i-1
          do 65 k=ne(i),ne(i)+ncp0-1
          if(lc.eq.1) then
          nc0=idiv
          else
          nc0=ncp
          end if
```

The parameters to udgrma are elements from the arrays xm, ym, xc, yc, udg, and the scalars ge and xnu. The xm and ym arrays, which are the boundary element coordinates, and the ge and xnu scalars, which are material properties, are invariant throughout the entire problem, and are placed into the tuple space by the Master routine at the beginning of the calculation. The xc and yc arrays are the crack path coordinates and are updated inside the "do 20 i =" loop. This fact requires the xc and yc arrays be placed into the tuple space prior to each execution of the

nested loop. The p_udgrma routine (written in C) is shown below. This routine simply places npc0 do_udgrma workers into the tuple space and retrieves the result data as it becomes available. The nested loop to retrieve the udgrma data has the nesting order reversed from the original FORTRAN code. This is because the udgrma data will become available in k order because we have npc0 processes performing k udgrma calculations. We note that this is a much finer grained parallel implementation than the Monte Carlo loop, and parallel efficiency will be highly dependent on the particular hardware and the compute/communicate ratio (it would be ideally suited for implementation on a hybrid-memory architecture).

```
void p_udgrma(double *xc, double *yc, double udg[2*nstpl*ncpl][2*nbp],
        int *ne, int *ncp0, int *n)
{
  int i, j, k;

  /* place xc and yc arrays into the TS */
  out("data-xc", xc);
  out("data-yc", yc);

    /* start npc0 do_udgrma processes */
    for (i=*ne; i< (*ne+*npc0); i++)
    {
       eval("udgrma", do_udgrma(i, *n));
    }

    /* retrieve the udgrma data */
    for (j=*ne; j<=(*ne+*ncp0-1); j++)
    {
       for (k=1; k<=n; k++)
       {
          in("udgrma_output", j, k, 1, ?udg[2*j-2][2*k-2]);
          in("udgrma_output", j, k, 2, ?udg[2*j-1][2*k-2]);
          in("udgrma_output", j, k, 3, ?udg[2*j-2][2*k-1]);
          in("udgrma_output", j, k, 4, ?udg[2*j-1][2*k-1]);
       }
    }
}
```

The do_udgrma routine shown below is the worker process for p_udgrma. It is responsible for retrieving the necessary data from the tuple space and calling the FORTRAN routine udgrma *n* times, and then outing the results from each call back to the tuple space. Each do_udgrma routine performs one iteration of the original outer loop, *i.e. j* is constant and unique for each do_udgrma node.

```
void do_udgrma(int j, int n)
{
  int i, k;
  double udg1, udg2, udg3, udg4;

  /* retrieve the xc, yc, xm, and ym arrays from the TS */
  rd("data-xc", ?xc);
  rd("data-yc", ?yc);
  rd("data-xm", ?xm);
  rd("data-ym", ?ym);
```

```
    for (k=1; k<=n; k++)
    {
        /* FORTRAN CALL */
        #ifdef PREPEND
        _udgrma(&xm[k-1],&ym[k-1],&xc[j-1],&yc[j-1],&xc[j],&yc[j],&udg1,&udg2,
                &udg3,&udg4);
        #elseif
        udgrma(&xm[k-1],&ym[k-1],&xc[j-1],&yc[j-1],&xc[j],&yc[j],&udg1,&udg2,
               &udg3,&udg4);
        #endif
        out("udgrma_output", j, k, 1, udg1);
        out("udgrma_output", j, k, 2, udg2);
        out("udgrma_output", j, k, 3, udg3);
        out("udgrma_output", j, k, 4, udg4);
    }
}
```

3.3.3 Parallel Performance of the Fatigue Reliability Problem with C-Linda

For this Phase I research we selected the virtual shared-memory programming approach for implementation and evaluation. There were three reasons for this. First the virtual shared-memory approach allows straightforward implementation of multiple levels of parallelism and, therefore, can be used to develop parallel control algorithms to automatically invoke multiple levels of parallelism (a proposed Phase II research task). Second, we believe that this approach has Phase III commercialization potential because it offers the advantage of portability across a diverse range of hardwares (including shared-memory, distributed-memory, and networks of workstations). Third, the virtual shared-memory approach will, in the future, allow easy adaptation of the software to hybrid-memory architectures, which are likely to be optimal for parallel PCM. We will evaluate here the efficiency of this approach for inplementing one level of parallelism. It remains for Phase II to evaluate the efficiency of this approach for implementing multiple levels of parallelism.

The fatigue reliability problem was parallelized using C-Linda as described above. The parallel performance was then tested on a distributed-memory network of workstations using the Monte-Carlo loop parallelism. The main reason for selecting the network platform was to determine the feasibility of using networks for parallel processing in addition to conventional parallel computers. The capability to use the network platform without modifying the parallel code, should significantly increase the chances for commercial success of this SBIR.

Figure 3-21 shows the near linear speedup obtained for the fatigue problem. Because the compute to communicate ratio is very large and because each processor has its own local memory, these systems exhibit scalable behavior (*i.e.*, near linear speedup as the number of processors increases). The excellent performance on a network with relatively small communications bandwidth indicates that high efficiency can be achieved on distributed-memory multiprocessors that are equipped with specialized communications hardware.

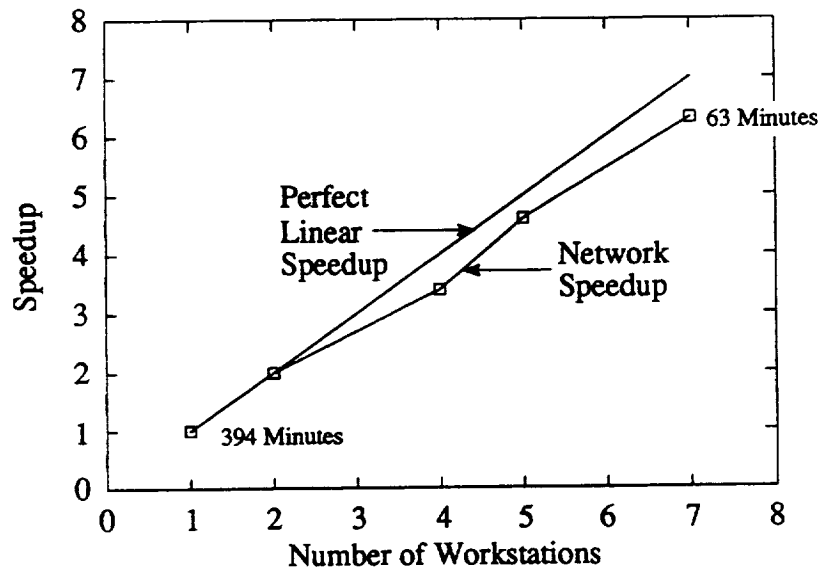


Figure 3-21. Parallel Speedup for Fatigue Reliability Problem on SPARCStation Network

CHAPTER 4

SUMMARY, CONCLUSIONS AND RECOMMENDATIONS

4.1 SUMMARY

The objective of this Phase I research was to establish the required software and hardware strategies to achieve large-scale parallelism in solving problems in probabilistic response analysis of high temperature composites. To meet this objective, several tasks were conducted. First, we identified the multiple levels of parallelism in probabilistic composite mechanics. Parallelism was identified in several areas including: (1) the probabilistic computations; (2) general structural mechanics; and (3) specialized parallelism in PCM. This research culminated in outlining a comprehensive multi-level computational strategy to exploit these sources of parallelism in a way that minimizes memory/processor requirements while minimizing parallel overhead. The strategy incorporates a top-down approach to maximize average granularity and specially designed computational algorithms (probabilistic substructuring coupled with domain decomposition and the stochastic preconditioned conjugate gradient solver). Parameters for determining how many levels of parallelism to invoke to maximize speedup for a particular problem and hardware platform were identified.

Next several software implementations and hardware efficiency investigations were undertaken to establish the most promising approaches to follow in Phase II and ultimately Phase III commercialization. The software implementations included a physical shared-memory model (for shared-memory computers), a message passing model (for distributed-memory computers), and a virtual shared-memory model (for either architecture, for hybrid architectures, and for networks of workstations). The hardware implementations included a shared-memory computer with twenty-four processors, and a distributed-memory network of seven workstations. The list below summarizes these studies:

1. ***Shared Memory, Alliant FX/2800.*** Two problems, one with small memory requirements (a 3D space truss) and one with larger memory requirements (fatigue life reliability of a plate with an initial defect via the stochastic boundary element method). The purpose here was to investigate the affect of memory requirements for parallel efficiency for shared memory systems. Parallel speedup studies were performed using one to twenty-four processors.

2. ***Distributed Memory Software Development using Message Passing Paradigm.*** Code was developed for solving the fatigue life reliability problem using CS Tools to investigate the feasibility of parallelizing probabilistic analysis using the message passing paradigm.

3. ***Distributed Memory Software Development using Virtual-Shared Memory Paradigm.*** Code was developed for solving the fatigue life reliability problem using C-Linda to investigate the feasibility of parallelizing multiple levels of parallelism using the virtual shared-memory paradigm. Both the Monte-Carlo simulation loop and computation of the influence coefficients during each simulation history were parallelized.

4. ***Distributed-Memory Network.*** The feasibility of achieving parallel speedup on a distributed-memory network of workstations using the virtual shared-memory programming paradigm was investigated. Parallel speedup studies were performed for the fatigue reliability problem using one to seven workstations.

4.2 CONCLUSIONS AND RECOMMENDATIONS

The investigations in Phase I demonstrated that it is possible to effectively parallelize probabilistic structural analysis codes to achieve high parallel speedup for certain classes of problems on certain hardware types. However, it is clear that special strategies will be needed to achieve large-scale parallelism to keep large numbers of processors busy and to treat problems with the large memory requirements encountered in practice.

Reducing memory/processor demand is a key factor to achieving large-scale speedup for both shared and distributed-memory platforms. For shared-memory, large memory requirements result in memory contention and congestion in processor-to-memory communications. In addition, when parallelizing only the top level probabilistic analysis computations, memory requirements increase almost proportionately with the number of processors. Hence, there is a potential that using all processors could result in memory demand that exceeds the size of physical memory, thereby requiring the use of secondary storage. In such cases speedup can reduce with increasing numbers of processors. For distributed-memory, we do not have the shared-memory bottleneck, however, if the task assigned to a processor cannot fit in its local memory its performance will be significantly slowed.

To alleviate memory/processor demand special computational strategies for parallel PCM are needed. A strategy was outlined herein that combines a domain decomposition approach with the probabilistic substructuring technique and the stochastic preconditioned conjugate gradient equation solver. A boundary element analog of the probabilistic substructuring technique was used successfully in solving the fatigue reliability example problem. The computational strategy reduces the memory demand per processor and also increases the degree of parallelism. We recommend, therefore, that research continue to investigate these special computational approaches.

Commercialization is a key objective of this SBIR. In this regard, we conclude that the parallel computing environment should be portable across a range of hardwares and should incorporate automated problem decomposition and parallelization control algorithms to free the user from this tedious task. The virtual shared-memory programming paradigm is well suited to meet the needs of portability and also provides the flexibility to implement the control algorithms and invoke the multiple levels of parallelism in PCM. In addition, using the virtual shared-memory approach will, in the future, enable the software to be ported to hybrid memory architectures that are now becoming available and are ideally suited for parallel solution of PCM problems. Finally, using the virtual shared-memory approach will allow for execution on widely available networks of single and multi-processor workstations, which will significantly increase the customer base. From the Phase I studies we can conclude that workstation networks are well suited to exploit at least the top level coarse-grained parallelism in PCM problems. It remains to

evaluate the efficiency of the virtual shared memory paradigm at exploiting multiple levels of parallelism.

Based on the Phase I studies we can summarize our conclusions and recommendations as follows:

1. There are several levels of parallelism in PCM problems that will need to be taken advantage of in order to fully exploit the potential of parallel processing computers.

2. Specially adapted computational algorithms should be developed for efficient parallel implementation of many practical problems in order to reduce memory requirements and processor idling.

3. Parallel control algorithms should be developed to automatically decompose a problem and exploit the multiple levels of parallelism in PCM problems to increase the practical usability of parallel PCM codes.

4. The parallel PCM code should be portable across a range of architectures to increase the commercial viability of the software. Availability on workstation networks is also desirable to further increase the customer base.

5. The virtual shared-memory programming paradigm can provide the desired portability and flexibility to easily exploit multiple levels of parallelism in PCM problems. It remains to eevaluate the efficiency of this approach in implementing multiple levels of parallelism.

6. Shared-memory hardware can be highly efficient for probabilistic analysis problems for small numbers of processors. Even for the large fatigue reliability problem, better than 90% efficiency was achieved for ten processors. However, parallel performance degrades with increasing numbers of processors and it is possible to obtain negative return with increasing numbers of processors (generally, because physical memory becomes overloaded such that disk paging is required).

7. Coding on shared-memory multi-processors is straightforward for a single level of parallelism. For multi-level parallelism, special constructs and significant code rewriting is required.

8. Distributed-memory architectures are preferable to shared-memory for achieving large-scale parallelism because PCM problems have at least one level of coarse-grained computations. Although distributed-memory systems have a disadvantage with regard to communications, the overhead cost associated with shared-memory is not justified. For a shared-memory machine, access to the shared memory will become a bottleneck when using large numbers of processors.

9. Hybrid-memory architectures, consisting of an interconnection of shared-memory processor nodes (*i.e.*, four to eight processors that share memory at a node) will likely be optimal for parallel PCM problems. This architecture maps directly to the multiple levels of both coarse and fine grained parallelism exhibited by PCM problems. This is an emerging technology and is typified by the Intel Paragon machine, networks of Silicon graphics multi-processor workstations, and the NASA Hypercluster machine.

CHAPTER 5

REFERENCES

Amdahl, G., "Validity of the Single Processor Approach to Achieving Large-Scale Computing Capabilities," *Proceedings of the Spring Joint Conference of AFIPS*, 1967.

Batchelor, G. K., and Green, J. T., 1972. "The Determination of the Bulk Stress in a Suspension of Spherical Particles to Order c^2," *Journal of Fluid Mechanics*, Volume 56, pp. 401-427.

Benveniste, Y., 1987. *Mech. Materials*, Volume 6, p. 147.

Bert, C. W., 1984. "A Critical Evaluation of New Plate Theories Applied to Laminated Composites, *Journal of Composite Structures*, Volume 2, pp. 329-347.

Budiansky, B., 1965. "On the Elastic Moduli of Some Heterogeneous Materials," *Journal of the Mechanics and Physics of Solids*, Volume 13, pp. 223-227.

Carriero, N., and Gelernter, D., 1989. *How to Write Parallel Programs: A Guide to the Perplexed*, Department of Computer Science, Yale University, New Haven, Connecticut.

Chamis, C. C., Stock, T. A., 1990. "Probabilistic Simulation of Uncertainties in Composite Uniaxial Strengths," *45th Annual Conf. Society of Plastics Industry Reinforced Plastics/ Composite Institute*, Washington, D. C., February 12-16.

Chamis, C. C., 1986a. *Computational Composite Mechanics for Aerospace Propulsion Structures*, NASA-TM-88965, NASA-Lewis, Cleveland, Ohio.

Chamis, C. C., 1986b. *Probabilistic Structural Analysis Methods for Space Propulsion System Components*, NASA-TM-88861, NASA-Lewis, Cleveland, Ohio.

Chamis, C. C., 1983. *Simplified Composite Micromechanics Equations for Hygral, Thermal, and Mechanical Properties*, NASA-TM-83320, NASA-Lewis, Cleveland, Ohio.

Chamis, C. C., and Sendeckyj, G. P., 1968. "Critique on Theories Predicting Thermoelastic Properties of Fibrous Composites," *Journal of Composite Materials*, Volume 2, Number 3, pp. 332-358.

Christensen, R. M., and Lo, K. H., 1986. *Journal of the Mechanics and Physics of Solids*, Volume 34, p. 639.

Christensen, R. M., and Lo, K. H., 1979. "Solutions for Effective Shear Properties in Three Phase Sphere and Cylinder Models," *Journal of the Mechanics and Physics of Solids*, Volume 27, pp. 315-330.

Christensen, R. M., 1979. *Mechanics of Composite Materials*, Wiley-Interscience, New York.

Dongarra, J. and Duff, I. S., 1992. "Advanced Architecture Computers," *Supercomputing in Engineering Analysis*, Ed H. Adeli, Marcel Dekker, Inc., New York.

Engelstad, S. P., and Reddy, J. N., 1991. "Nonlinear Probabilistic FEM for Composite Shells," *ASCE Engineering Mechanics Specialty Conference on Mechanics Computing in the 1990's and Beyond*, Columbus, Ohio, pp. 173-177, 20-22 May.

Erdogan, F., and Sih, G. C., 1963. "On the Crack Extension in Plates Under Plane Loading and Transverse Shear," *Journal of Basic Engineering*, ASME, Volume 85, pp. 519-527.

Eshelby, J. D., 1957. "The Determination of the Field of an Ellipsoidal Inclusion and Related Problems," Proceedings of the Royal Society, London, Volume 4, Number 241, pp. 376-396.

Farhat, C., 1992. "Finite Element Analysis on Concurrent Machines," Chapter 7 in, *Parallel Processing in Computational Mechanics*, edited by H. Adeli, Marcel Dekker, New York.

Farhat, C., 1991. "A Lagrange Multiplier-Based Divide-and-Conquer Finite Element Algorithm," *Computing Systems in Engineering*, Volume 2, No. 2/3, pp. 149-156.

Flynn, M. J., 1966.; "Very High-Speed Computing Systems," *Proceedings of IEEE*, Volume 54, Number 12, pp. 1901-1909, December.

Frank, R. A., *et al.*, 1991. *Feasibility Investigation for Development of an Organiz Matrix Composite (OMC) Heat Sink for Electronic Devices*, Applied Research Associates, Final Report 5624, Naval Weapon Support Center, Crane, Indiana, 14 February.

Green, A. E., and Naghdi, P. M., 1982. "A Theory of Laminated Composite Plates," *IMA Journal of Applied Mathematics*, Volume 29, pp. 1-23.

Halpin, J. C., and Tsai, S. W., 1967. *Environmental Factors Estimation in Composite Materials Design*, AFML-TR-67-423.

Hashin, Z., 1983. "Analysis of Composite Materials — A Survey," *Journal of Applied Mechanics*, Volume 50, pp. 481-505.

Hashin, Z., 1972. *Theory of Fiber-Reinforced Materials*, NASA CR-1974.

Hashin, Z., 1962. ""The Elastic Moduli of Heterogeneous Materials," *ASME Journal of Applied Mechanics*, Volume 29, pp. 143-150.

Hashin, Z., 1959. "The Moduli of an Elastic Solid Containing Spherical Particles of Another Elastic Material," *Nonhomogeneity in Elasticity and Plasticity*, Ed. W. Olszak, Pergamon Press, pp. 463-478.

Hill, R., 1952. "The Elastic Behaviour of a Crystalline Aggregate," *Proceedings of the Physical Society*, Volume A65, pp. 349-354.

Hopkins, D. A., and Chamis, C. C., 1985. *A Unique Set of Micromechanics Equations for High Temperature Metal Matrix Composites*, NASA-N86-24757.

Jones, R. M., 1975. *Mechanics of Composite Materials*, Hemisphere Publishing Corporation, New York.

Komzsik, L., and Rose, T., 1991. "Substructuring in MSC/NASTRAN for Large-Scale Parallel Applications," *Computing Systems in Engineering*, Volume 2, Number 2/3, pp. 167-173.

Lin, T. H., 1968. *Theory of Inelastic Structures*, John Wiley and Sons, Inc.

Lua, Y. J., Liu, W. K., and Belytschko, T., 1992a. "A Stochastic Damage Model for the Rupture Prediction of a Multi-Phase Solid. Part I: Parametric Studies," *International Journal of Fracture Mechanics*, Volume 55, pp. 321-340.

Lua, Y. J., Liu, W. K., and Belytschko, T., 1992b. "A Stochastic Damage Model for the Rupture Predication of a Multi-Phase Solid. Part II: Statistical Approach," *International Journal of Fracture Mechanics*, Volume 55, pp. 341-361.

Lua, Y. J., Liu, W. K., and Belytschko, T., 1992c. "Life Prediction of a Curvilinear Fatigue Crack Growth by SBEM," *ASME 1992 Annual Winter Symposium on Reliability for Mechanical System Design*, Anaheim, California, 8-13 November.

Lua, Y. J., Liu, W. K., and Belytschko, T., 1992d. "A Stochastic Approach to the Fatigue Growth Reliability," *Proceedings of the Specialty Conference on Probabilistic Mechanics and Structural and Geotechnical Reliability*, Denver, Colorado, 8-10 July.

Mase, G. T., Murthy, P. L. N., and Chamis, C. C., 1991. *Probabilistic Micromechanics and Macromechanics of Polymer Matrix Composites*, NASA Technical Memorandum 103669, prepared for the 14th Annual Energy Sources Technology Conference, American Society of Mechanical Engineers, Houston, Texas, 20-24 January.

Miki, M., Murotsu, Y., Murayama, N., and Tanaka, T., 1992. "Application of Lamination Parameters to the Reliability-Based Stiffness Design of Fibrous Laminated Composites," *Proceedings of the 33rd AIAA Structures, Structural Dynamics, and Materials Conference*, April 13-15, Dallas, 1992.

Mura, T., 1982. *Micromechanics of Defects in Solids*, Martinus Nijhoff, The Hague.

Murakami, H., 1986. "Laminated Composite Plate Theory with Improved In-Plane Responses," *Journal of Applied Mechanics*, Volume 53, pp. 661- 666.

Norris, A. N., 1985. *Mech. Materials* Volume 4, p. 1.

Pagano, N. J., and Soni, S. R., 1983. "Global-Local Laminate Variational Model," *International Journal of Solids Structures*, Volume 19, pp. 207-228.

Paris, P. C., and Erdogan, F., 1963. "A Critical Analysis of Crack Propagation Laws," *Journal of Basic Engineering*, ASME, Volume 85, pp. 528-534.

Reddy, J. N., 1984. "A Simple Higher-Order Theory for Laminated Plates," *Journal of Applied Mechanics*, Volume 51, pp. 745-752.

Reissner, E., and Stavsky, Y., 1961. "Bending and Stretching of Certain Types of Heterogeneous Aelotropic Elastic Plates," *Journal of Applied Mechanics*, Volume 28, pp. 402-408.

Reuss, A., 1929. "Berechnng der Fliessgrense von Mischkristallen auf Grund der Plastizitätsbedingung für Einkristalle," *Zeitschrift fur Angewandte Mathematik und Mechanik*, Volume 9, pp. 49-58.

Seide, P., 1980. "An Improved Approximate Theory for the Bending of Laminated Plates," *Mechanics Today*, Ed. S. Nemat-Nasser, Volume 50, pp. 451-466.

Srinivas, S., and Rao, A. K., 1970. "Bending, Vibration, and Buckling of Simply Supported Thick Orthotropic Rectangular Plates and Laminates," *International Journal of Solids and Structures*, Volume 6, pp. 463-481.

Stock, T. A., Bellini, P. X., Murthy, P. L., and Chamis, C. C., 1988. "Probabilistic Composite Micromechanics," *29th Structures, Structural Dynamics, and Materials Conference*, Part 3, AIAA, New York, pp. 1289-1298.

Sues, R. H., Chen, H-C., Twisdale, L. A., Chamis, C. C., and Murthy, P. L. N., 1991a. "Programming Probabilistic Structural Analyses for Parallel Processing Computers," *Proceedings of the AIAA/ASME/ASCE/AHS/ASC 32nd SDM Conference*, Baltimore, Maryland, 6-8 April.

Sues, R. H., Chen, H-C., and Twisdale, L. A., 1991b. *Probabilistic Structural Mechanics Research for Parallel Processing Computers*, NASA CR-187162, August.

Taya, M., and Arsenault, R. J., 1989. *Metal Matrix Composites*, Pergamon Press, Oxford.

Thanedar, P. B., Chamis, C. C., 1990. *Composite Laminate Tailoring with Probabilistic Constraints and Loads*, NASA Technical Memorandum 102515, prepared for the Energy Technology Conference and Exhibition, sponsored by the American Society of Mechanical Engineers, New Orleans, Louisiana, 14-18 January.

Voigt, W., 1910. *Lehrbuch der Kristallphysik*, Teubner, Berlin.

Whitney, J. M., and Sun, C. T., 1973. "A Higher Order Theory for Extensional Motion of Laminated Composites," *Journal of Sound Vibrations*, Volume 30, pp. 85-97.

Whitney, J. M., and Pagano, N. J., 1970. "Shear Deformation in Heterogeneous Anisotropic Plates," *Journal of Applied Mechanics*, Volume 37, pp. 1031-1036.

Wu, T. T., 1966. "The Effect of Inclusion Shape on the Elastic Moduli of a Two-Phase Material," *International Journal of Solids and Structures*, Volume 2, pp. 1-8.

Yang, P. C., Norris, C. H., and Stavsky, Y., 1966. "Elastic Wave Propagation in Heterogeneous Plates," *International Journal of Solids and Structures*, Volume 2, pp. 665-684.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (*Leave blank*)

2. REPORT DATE
March 1994

3. REPORT TYPE AND DATES COVERED
Final Contractor Report

4. TITLE AND SUBTITLE
Parallel Computing for Probabilistic Response Analysis of High Temperature Composites

5. FUNDING NUMBERS
WU–505–63–5B
C–NAS3–26576

6. AUTHOR(S)
R.H. Sues, Y.J. Lua, and M.D. Smith

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Applied Research Associates, Inc.
4300 San Mateo Blvd., NE, Suite A220
Albuquerque, New Mexico 87110

8. PERFORMING ORGANIZATION REPORT NUMBER
E–8550

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Lewis Research Center
Cleveland, Ohio 44135–3191

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
NASA CR–194467

11. SUPPLEMENTARY NOTES
Project Manager, Christos C. Chamis, Structures Division, organization code 5200, NASA Lewis Research Center, (216) 433–3252.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified - Unlimited
Subject Category 39

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 words)***
The objective of this Phase I research was to establish the required software and hardware strategies to achieve large-scale parallelism in solving PCM problems. To meet this objective, several investigations were conducted. First, we identified the multiple levels of parallelism in PCM and the computational strategies to exploit these parallelisms. Next, several software and hardware efficiency investigations were conducted. These involved the use of three different parallel programming paradigms and solution of two examples problems on both a shared-memory multiprocessor and a distributed-memory network of workstations.

14. SUBJECT TERMS
Parallelization; Paradigms; Efficiency methods; Granular configurations; Multi-level parallelism; Shared-memory; Distributed memory; Example problems

15. NUMBER OF PAGES
60

16. PRICE CODE
A04

17. SECURITY CLASSIFICATION OF REPORT
Unclassified

18. SECURITY CLASSIFICATION OF THIS PAGE
Unclassified

19. SECURITY CLASSIFICATION OF ABSTRACT
Unclassified

20. LIMITATION OF ABSTRACT