NASA Contractor Report 195312

Portable Parallel Stochastic Optimization for the Design of Aeropropulsion Components

Robert H. Sues and G.S. Rhodes
Applied Research Associates, Inc.
Raleigh, North Carolina

October 1994

Prepared for
Lewis Research Center
Under Contract NAS3–26839

National Aeronautics and
Space Administration

TABLE OF CONTENTS

Chapter 1 Introduction 1-1
1.1 Background 1-1
1.2 Objectives and Scope 1-1

Chapter 2 Portable Parallel Stochastic Optimization 2-1
2.1 Introduction 2-1
2.2 Background on Parallel Processing and Portable Parallel Programming 2-1
2.3 Parallel Stochastic Optimization 2-5
2.3.1 General Formulation and Review 2-5
2.3.2 Formulation for Parallel Multi-Disciplinary Stochastic Optimization 2-7
2.4 Parallelism in Multi-Disciplinary Stochastic Optimization 2-12
2.4.1 Sources of Parallelism in Coupled Aeromechanical Design 2-12
2.4.2 Computational Strategy for Multi-Level Parallelism 2-14
2.4.3 Special-Purpose Computational Algorithms 2-17

Chapter 3 Parallel Implementation Using Parallel Virtual Machine (PVM) 3-1
3.1 Introduction 3-1
3.1.1 ANSI Standard Programming Languages 3-1
3.1.2 Portable PVM 3-1
3.2 Parallel Computer Systems 3-2
3.3 Advanced Propfan Blade Demonstration Problem 3-3
3.3.1 Problem Description 3-3
3.3.2 Brief Description of the VORP Aerodynamics Code 3-9
3.4 Implementation of the MSO 3-12
3.4.1 Overview 3-12
3.4.2 Parallelization of the Demonstration Problem 3-13
3.5 Parallel Performance 3-20
3.5.1 Description of the Timing Studies 3-20
3.5.2 Phase I Results 3-22

Chapter 4 Summary, Conclusions, and Recommendations 4-1
4.1 Summary 4-1
4.2 Conclusions and Recommendations 4-2

References R-1

LIST OF FIGURES

Figure 2-1 Parallel Architectures—Memory Taxonomy 2-2

Figure 2-2 Solution Algorithm for Stochastic Optimization 2-8

Figure 2-3 Top-Down Strategy for Parallel MSO Problem Decomposition 2-16

Figure 2-4 Probabilistic Substructuring 2-17

Figure 3-1 Hypercube Architecture 3-3

Figure 3-2 Increased Propulsive Efficiency with High-Speed Turboprops 3-4

Figure 3-3 Typical Advanced Propfan Blade Experimental Model 3-7

Figure 3-4 Representation of the Flow Field in the VORP Code 3-12

Figure 3-5 Velocity Influence Coefficient of an Element due to Itself and due to Neighboring Elements 3-15

Figure 3-6 Typical Propfan Blade and Loads 3-22

Figure 3-7 Thrust Versus Advance Ratio 3-24

Figure 3-8 Propeller Efficiency Versus Advance Ratio 3-24

Figure 3-9 Blade Twist Distributions 3-25

Figure 3-10 Parallel Speedup for Sensitivity Coefficient Calculations on the Intel 3-26

Figure 3-11 Theoretical Maximum Parallel Efficiency for Single-Level Parallelism 3-27

Figure 3-12 Advanced Turboprop Blade Differential Surface Pressures 3-27

Figure 3-13 Parallel Speedup for Influence Coefficient Calculations on the Intel 3-29

Figure 3-14 Parallel Efficiency for Influence Coefficient Calculations on the Intel 3-30

Figure 3-15 Parallel Speedup for Sensitivity Coefficient Calculations on the IBM RS/6000 Network 3-30

Figure 3-16 Speedup Results for Multi-Level Parallelism on the Intel: PSC/860 (Ten Clusters Used for Each Multi-Level Study) 3-32

Figure 3-17 Optimized Cold Shape Twist Distribution 3-33

Figure 3-18 Deformed "Hot" Shape 3-35

Figure 3-19 Cumulative Distribution Function for Tip Displacement in the Z-Direction 3-36

LIST OF TABLES

Table 2-1 Parallel-Programming Toolkits 2-3

Table 2-2 Sources of Parallelism in Probabilistic Mechanics 2-13

Table 3-1 Statistic Parameters for Stochastic Analysis of the Propfan Blade 3-36

Table 3-2 Response Statistics 3-36

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

The continuing rapid advancement of computer technology plays an important role in the evolution of engineering and mechanical design. Today, as in the past, engineers often use a "trial and error" approach to design, using their experience as a guide. Although trial and error design has its place, this approach is especially inefficient when dealing with complex design problems which involve many different engineering disciplines, and for problems, such as the design of high-speed-civil-transport vehicles or the design of advanced propulsion systems, which involve new technologies for which little practical experience exists. It is in design problems such as these that high-power computers are becoming more and more useful. With advanced analysis and numerical optimization techniques, engineers and scientists are attempting to eliminate the expensive man-in-the-loop iterations, and solve multi-disciplinary design problems completely by computer. This emerging technology is often referred to as multi-disciplinary design optimization.

The computational cost of these multi-disciplinary optimizations can be quite large. Many complex calculations must be performed repeatedly within an optimization loop. These tasks can often become unfeasible on a single computer. The advent of parallel and distributed computing and continuous advances in computing technology have spawned new potential for reducing the computer time required for multi-disciplinary design optimizations. Problems which cannot be solved on a single computer can now be solved using a network of several computers.

By using parallel and distributed computing, engineers can consider aspects of a design which were previously ignored. For example, the treatment of uncertainties in design optimization has long been recognized as important. Both design and constraint variables with a large degree of uncertainty can significantly affect the optimum design, and may govern the design constraints. In addition, in pushing performance limits it is crucial that aircraft reliability be quantified. Probabilistic methods are a growing part of optimized design, and they can be efficiently used in a parallel computing environment. The combination of multi-disciplinary stochastic optimization (MSO) and parallel computing promises to be an innovative solution to optimization of advanced aerospace systems design.

1.2 OBJECTIVES AND SCOPE

The primary goal of this research program is to develop a methodology for performing parallel Multi-Disciplinary Stochastic Optimization (MSO) of aerospace systems. The MSO methodology will be implemented in a portable programming environment, meaning the source code can be compiled and executed on many different computer systems ranging from massively parallel super-computers to networks of

engineering workstations. Ideally, the code should be able to execute on a network of heterogeneous computers.

A complimentary goal is to achieve large-scale parallelism in solving MSO problems. 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 software/hardware 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 represents 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 portable parallel programming approaches that meet our requirements for the portability, extensibility and efficiency necessary to achieve commercial success.

2. Demonstrate source code portability by executing an example application on a massively parallel distributed memory system and on a network of workstations.

3. Demonstrate feasibility of achieving high parallel efficiency and large scale parallelism via parallel timing studies.

4. Evaluate lessons learned from the example computations and 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 three chapters. In Chapter 2, we present the results of our research to identify a portable parallel programming approach that meets our requirements, a newly formulated MSO methodology, identify the multiple levels of parallelism, and discuss strategies for exploiting this parallelism. In Chapter 3, we present the implementation of the MSO methodology in a portable parallel programming environment. We also present results from an example problem, the shape optimization of an advanced propfan blade. The example was executed on a 128-node Intel iPSC/860 hypercube, a network of IBM RISC/6000 workstations, and a single Hewlett-Packard Apollo 9000/730 workstation. The list below summarizes these studies.

1. ***Parallel Optimization, Intel iPSC/860.*** Parallel computation of sensitivity coefficients used in aerodynamic shape optimization of an advanced

propfan blade. This first study is used as a benchmark and to confirm expectations of high parallel efficiency for coarse-grained analyses, using from one to twenty processors.

2. ***Parallel Aerodynamic Analysis, Intel iPSC/860.*** Parallel computation of aerodynamic influence coefficients to obtain loads on the propfan blade. This study investigates the feasibility of achieving high parallel efficiency for a finer-grained problem. Analyses are executed using from one to fifty processors.

3. ***Parallel Optimization, IBM RS/6000 workstation network.*** Repeat of study described under item 1 for the workstation network. Here we investigate the portability of the PVM toolkit and study parallel efficiency over a workstation network, both in a dedicated mode and in normal operation mode, using from one to twenty workstations.

4. ***Multi-level Parallelism, Intel iPSC/860.*** Simultaneous parallel computation of both sensitivity coefficients and influence coefficients. This study investigates the feasibility of simultaneously exploiting more than one level of parallelism (which will be necessary for achieving large scale speedup for practical problems of interest). We use a top-down approach and exploit the coarsest grained part of the problem first (the sensitivity coefficients) and use remaining available processors for the finer grained part of the problem (the influence coefficients). Analyses are executed using from ten to forty processors.

5. ***Multi-disciplinary Optimization, HP 9000/730 Workstation.*** Coupled aeromechanical optimization of the advanced propfan blade. An improved optimization procedure is made possible since the blade shape can be optimized starting from the cold shape as opposed to a presumed hot shape. The purpose of this study is to investigate the feasibility of performing multi-disciplinary optimization, determine computational resources required, and to identify special requirements that will be needed for parallelization in Phase II.

6. ***Stochastic Analysis HP, 9000/730 Workstation.*** Stochastic structural analysis of the propfan blade under load was executed. Parallelization was not performed since the feasibility of parallel probabilistic analysis has been demonstrated in earlier research. The purpose was to demonstrate the feasibility of stochastic analysis for the example problem.

In Chapter 4, we present our conclusions and recommendations.

CHAPTER 2

PORTABLE PARALLEL STOCHASTIC OPTIMIZATION

2.1 INTRODUCTION

Multi-disciplinary Stochastic Optimization (MSO) problems are computationally expensive. Fortunately, however, these problems have many levels of inherent parallelism. In this chapter we present the stochastic optimization methodology and identify sources of parallelism. With improvements in processor speed, inter-processor communication, and high-speed mass storage capabilities, it is now possible to perform MSO of key aircraft aeropropulsion components, combining at least Euler code aerodynamics and nonlinear structural mechanics. Many corporations have the computational power today. The missing key piece is the software methodology to perform MSO efficiently and in an automated fashion, so that the engineer need not be burdened with extensive parallel coding. The stochastic optimization methodology formulated in this research is designed to be able to effectively take advantage of emerging parallel hardware and workstation networks.

2.2 BACKGROUND ON PARALLEL PROCESSING AND PORTABLE PARALLEL PROGRAMMING

The purpose in using parallel processing is to reduce the time required to complete a computation by dividing the task into subtasks and executing many subtasks simultaneously. The cost of doing parallel processing in the past was quite expensive. However, in recent years, computer component costs have dropped steadily, as power has increased. Currently, single-processor desktop computers are available which have more computational power than mainframe computers less than 10 years old. This low cost per component and cost/performance ratio is making parallel computing a practical reality, whether as a massively parallel supercomputer or as a network of workstations. It is common even for small companies to have several 32-bit or 64-bit RISC processor workstations connected on a Local Area Network. Since information can be passed between the workstations, it is possible to distribute tasks to the different workstations, which execute their tasks in parallel.

This section provides information which is relevant to the current research. Details of the principle ideas in parallel processing can be found in Sues et al. [1991a, b; 1992]. A survey of the range of parallel architectures is given by Dongarra and Duff [1992].

The workstation network provides parallel processing capabilities at the low end. At the high-end are the multiple-processor parallel computers. There are several ways to classify parallel architectures. For our purposes, a memory-based taxonomy is most appropriate. We divide parallel processing hardware into three groups as shown in Figure 2-1. It is desirable for the purposes of this research to support all three memory architectures in a fashion which is portable; that is, the source code should be identical regardless of the hardware platform or memory architecture. This portability is

acquired by using parallel-programming toolkits that disguise hardware and architecture-dependent code with common subroutine calls that do not change from machine to machine.

In addition to providing portability for parallel programming, these parallel-programming toolkits often introduce useful paradigms which aid program development. In much the same way as the Object-Oriented-Programming paradigm simplifies the development of large software projects in general, parallel programming paradigms simplify the distribution of subtasks for parallel program development. Table 2-1 summarizes a variety of parallel-programming toolkits.

In the table, there are three different programming paradigms. The message-passing paradigm is the conventional approach to parallel programming. In fact, most UNIX-based operating systems provide native, hardware-specific, function calls for performing message-passing. All implementations of message-passing provide nearly identical capabilities; thus, a program which is written using one message-passing toolkit can easily be ported to another message-passing toolkit. For this reason, when choosing a message-passing toolkit, the primary concerns are those of hardware support, stability and reliability, and acceptance in the community of hardware and software manufacturers.

The virtual-shared-memory paradigm of the Linda parallel programming language is quite different from message-passing. Linda provides a mechanism for storing data structures which can be accessed by any processor, whether it be local or on a network. In Linda, these data structures are known as "tuples," and they are stored in a "tuple-space" which represents the virtual shared memory [Carriero and Gelernter,

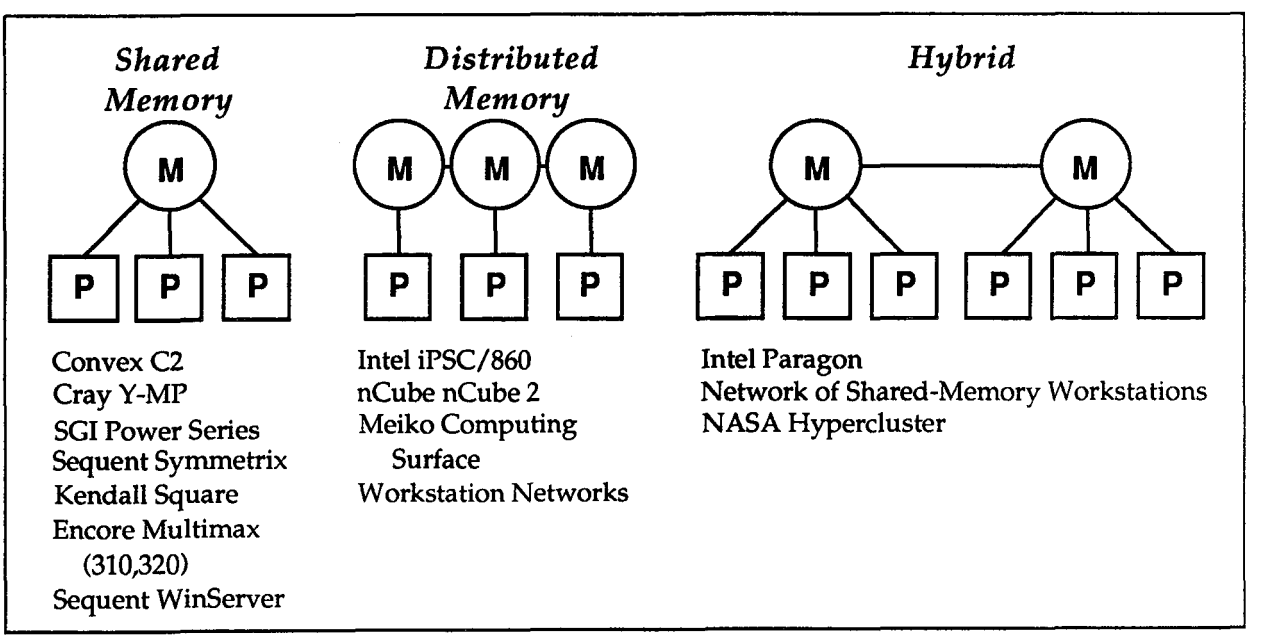

Figure 2-1. Parallel Architectures—Memory taxonomy

1989]. The tuple-space memory model of Linda simplifies the parallel programming task considerably and is portable, but introduces a communication overhead cost. Since the tuples can be stored in the local memory of any node, a search for a tuple in general requires cross-node communication. This is especially true for large problems where most of the local memory is required for data storage, and little memory is left for task maintenance.

The aggregate distributed objects approach to parallel programming, as implemented using the Interwork II toolkit [Bain, 1990], is similar to Linda in that it distributes data objects across the nodes in such a fashion that they may be accessed by any node without explicitly transmitting and receiving messages containing the data. This method can be considered a superset of the tuple-space concept which allows the programmer to define a multi-level hierarchy of global, distributed data objects. The method potentially introduces the same performance losses as Linda.

TABLE 2-1. PARALLEL-PROGRAMMING TOOLKITS

TOOLKITS	PARADIGM	PLATFORMS	PROS	CONS
Express	Message-Passing	Workstation Network, MPP	commercial toolkit	not widely used
Linda	Virtual-Shared-Memory	Workstation Network, MPP	commercial toolkit, widely used, heterogeneous hardware	potential performance bottleneck
Isis	Message-Passing	Workstation Network	public-domain and commercial versions, heterogeneous hardware	not widely used
PVM 3.1	Message-Passing	Workstation Network, MPP	supports many platforms, widely used, heterogeneous hardware, widely accepted	not commercially supported (continuing research project at ORNL)
Interwork II	Message-Passing, Aggregate-Distributed-Objects	iPSC/860, iPSC/2	commercial toolkit, close ties with Intel	supports only Intel hypercube platforms
APPL	Message-Passing	Workstation Network, MPP	heterogeneous hardware	experimental
PAX-2	Message-Passing	Workstation Network	based on PVM	does not support MPP yet

MPP=Multiple-Parallel-Processors

The Parallel Virtual Machine (PVM) toolkit is a portable message-passing toolkit which has been developed by the Oak Ridge National Laboratory (ORNL) [ORNL, 1993]. It is under continuous development at ORNL, Emery University, the University of Tennessee, and Carnegie Mellon University. PVM is fully public domain, and is one of the more popular portable parallel toolkits. The current version, PVM 3.1, will work on networks of workstations, even heterogeneous networks. In order to handle multiple-sized integer and floating point variables, and to handle different byte-orderings, PVM uses a structured message "packing" and "unpacking" methodology. For example, while one processor might store a two-byte integer value with the least-significant byte at a lower address than the most-significant byte, other processors store the integer with the bytes reversed. Rather than require the programmer to be knowledgeable about intricate details such as byte-ordering, PVM uses its packing/unpacking methodology to automatically order the bytes as appropriate for a particular architecture. Apart from this, PVM is a no-frills toolkit. It provides all the features and functions which are necessary, but does not introduce special-purpose features which would introduce a larger performance drop from the native operating system functions.

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

$$S_N = \frac{1}{\alpha + (1-\alpha)/N + f(N)} \tag{2-1}$$

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

$$\varepsilon = \frac{S_{N,obs}}{S^T} x100 \tag{2-2}$$

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 2-1 reduces to Amdahl's law [Amdahl 1967]).

The overhead, $f(N)$, will depend on how well the processor work load is balanced (to prevent idling), the amount of interprocessor communication required, and general concurrency management required. These issues are well known and well documented and we do not go into detail here (Sues et al. 1991b). However, the concept of granularity, which relates to interprocessor communication is particularly germane to this study and we provide some background here.

Granularity is a function of the compute to communicate ratio or the number of code steps which are executed on a given processor divided by the total parallel

communication time for that processor. $f(N)$ is therefore inversely related to granularity, so a problem with extremely coarse or large granularity will have small values of $f(N)$, in the absence of other sources of overhead. Problems are generally divided into classes of granularity: "fine" granularity contributes to high values of $f(N)$ and lower parallel efficiency, "coarse" granularity yields low values of $f(N)$ and higher efficiency, and "medium" granularity is in between. A general description of granularity has been given which defines "very fine grained" problems as those which execute 0 to 16 steps before requiring additional parallel input, " fine-grained" problems execute 17 to 256 steps before requiring parallel input, " medium-grained" problems execute 257 to 4096 steps before requiring additional input, and "coarse-grained" problems execute more than 4097 steps before requiring parallel input [Spector 1981]. This definition varies depending on the size of the parallel data packets. Medium to coarse-grained problems are suitable for networks of workstations, while fine-grained problems are not.

2.3 PARALLEL STOCHASTIC OPTIMIZATION

Stochastic optimization problems are computationally intensive because they require evaluation of response sensitivity coefficients at each design iteration. In general, however, the sensitivity information required in checking the reliability constraints can also be used in the optimization procedure. Hence, while the stochastic optimization will be more computationally intensive than deterministic optimization, the computational effort of stochastic optimization can be on the order of stochastic analysis. In addition, much of the effort can be performed in parallel. In the following we present a brief review of stochastic optimization and then a specific formulation developed under this Phase I research.

2.3.1 General Formulation and Review

The general formulation for the stochastic optimization problem is as follows:

Min (or Max) $f(x), x \in R^n$

Subject to: $g_j(x)=0$, for $j=1,M_e$

$$g_j(x) \geq 0, for\ j = M_e + 1, ...M$$

$$x_l \leq x \leq x_u$$

where $f(x)$ is the objective function and $g_j(x)$ are the constraint functions. The first M_e constraints are equality constraints and the remaining M-M_e constraints are inequality constraints.

As an example, for the design of a propfan blade, the objective function $f(x)$ might be the thrust provided by the blade, which is a function of the blade geometry

and the operating conditions. The geometric design variables, given suitable airfoil and chord distributions, are the radial variation of the blade pitch angle (twist) and of the blade sweep angle. The constraints include the practical limits on the design variables, pitch angle, blade tip sweep angle, an available engine power constraint or a fuel consumption rate constraint, and static and dynamic aeroelastic performance criteria. Since the aeroelastic response and lifetime performance of the blade will be a function of the mechanical properties, the loadings, and the continuous micro and macro-mechanical damage processes that can occur over the service life of the blade; the structural behavior will be subject to significant uncertainty, and hence the aeroelastic performance criterion constraints should properly be treated non-deterministically. That is, we require that the aeroelastic performance criteria meet or exceed a specified reliability goal, (e.g., the probability that the blade will not experience fatigue failure or fracture within a reasonable aircraft lifetime is greater than 95%).

A number of optimization algorithms have been used to solve stochastic optimization problems. For this research we have selected the sequential quadratic programming algorithm. This algorithm has been successfully used herein for optimizing the aerodynamics characteristics of propeller blades (see Chapter 3.0) and has recently been applied in stochastic structural optimization problems by Mahadevan [1992]. The method is based on an iterative formulation and solution of quadratic programming subproblems. The method obtains subproblems by using a quadratic approximation of the Langrangian and by linearizing the constraints. That is,

$$min\ \frac{1}{2}d^T B_k d + \nabla f(x_k)T_d\,,\ x \in R^n$$

subject to: $\nabla g_j(x_k)^T d + g_j(x_k) = 0, j = 1,\ldots, Me$

$$\nabla g_j(x_k)^T d + g_j(x_k) \geq 0, j = M_e+1,\ \ldots,\ M$$

$$x_l - x_u \leq d\, x_u - x_u$$

where $f(x)$ is the design objective function given above, B_k is a positive definite approximation of the Hessian matrix, and x_k is the solution at the current iteration. Letting d_k be the solution of the subproblem, a line search is used to find the new vector of design variables x_{k+1} as

$$x_{k+1} = x_k + \lambda d_k,\ \lambda \in [0,1]$$

such that a merit function will have a lower function value (higher for maximization) at the new design point. The matrix B_k can be updated using the Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm.

The reliability-based constraints are evaluated by using well established structural reliability concepts and can be stated as:

$$\beta_i \geq \beta_l^{'}, i = 1, M$$

where β_i is the reliability index for the ith performance criteria and the lower bound $\beta_l^{'}$ specifies the minimum required values for the reliability index. The reliability index is a relative measure of the probability that the component response (e.g., blade flutter frequency, deflection at a point, etc.) will not exceed desired limits and accounts for all uncertainties in computing the response.

The reliability index β is obtained as

$$\beta = (y^{*T} y^{*})^{1/2}$$

where y^{*} is the point of minimum distance from the origin of the limit state G(Y)=0 (where G(Y) is the performance function limit state, e.g., $G(Y)=\delta_1 - \delta_{1u} \leq 0.0$, where δ_l is the blade deflection at a point and δ_{1u} is the performance limit); and Y is the vector of uncorrelated standard normal variables. The transformation from X to Y can be achieved by using the Nataf-model transformation, proposed by Nataf [1962] and enhanced by Liu and DerKiureghian [1986].

Solution for the reliability index β can be by advanced first or second order reliability methods known in the literature as FORM/SORM or fast probability integration [e.g., Rackwitz and Fiessler 1978; Madsen et al. 1986; Twisdale, Sues, and Murphy 1988; Wu 1987], by response surface methods [Schueller 1989], by Monte Carlo Simulation (MCS) with variance reduction (importance sampling or stratified sampling), or by direct Monte Carlo Simulation. Hybrid methods such as the combined response surface and FORM method are also sometimes used. Note that when simulation methods are used we compute probabilities explicitly rather than use the reliability index β.

2.3.2 Formulation for Parallel Multi-Disciplinary Stochastic Optimization

Under this Phase I study, we have formulated a stochastic optimization approach that builds on earlier research in stochastic optimization and lends itself well to parallelization. This approach, which will be fully implemented in Phase II, recognizes that the most computationally efficient algorithm will be a function of the computing hardware and therefore, incorporates two algorithm paths, an MCS path and a FORM/SORM path. The approach was formulated to meet the following requirements/specifications: (1) parallelizable, (2) applicable for different numbers of each type of random variable (state and decision) (3) ability to treat large problems with many variables, (4) ability to handle large numbers of nonlinear constraints, and (5) generally applicable to the class of problems of interest. The formulation is depicted in Figure 2-2.

As mentioned two different probabilistic analysis paths are provided, since the most computationally expedient method is both problem and hardware specific. We

will briefly describe each path and then provide measures of the computational effort of each path. For either path we begin with an initial estimate of all of the design variables, just as in any design problem.

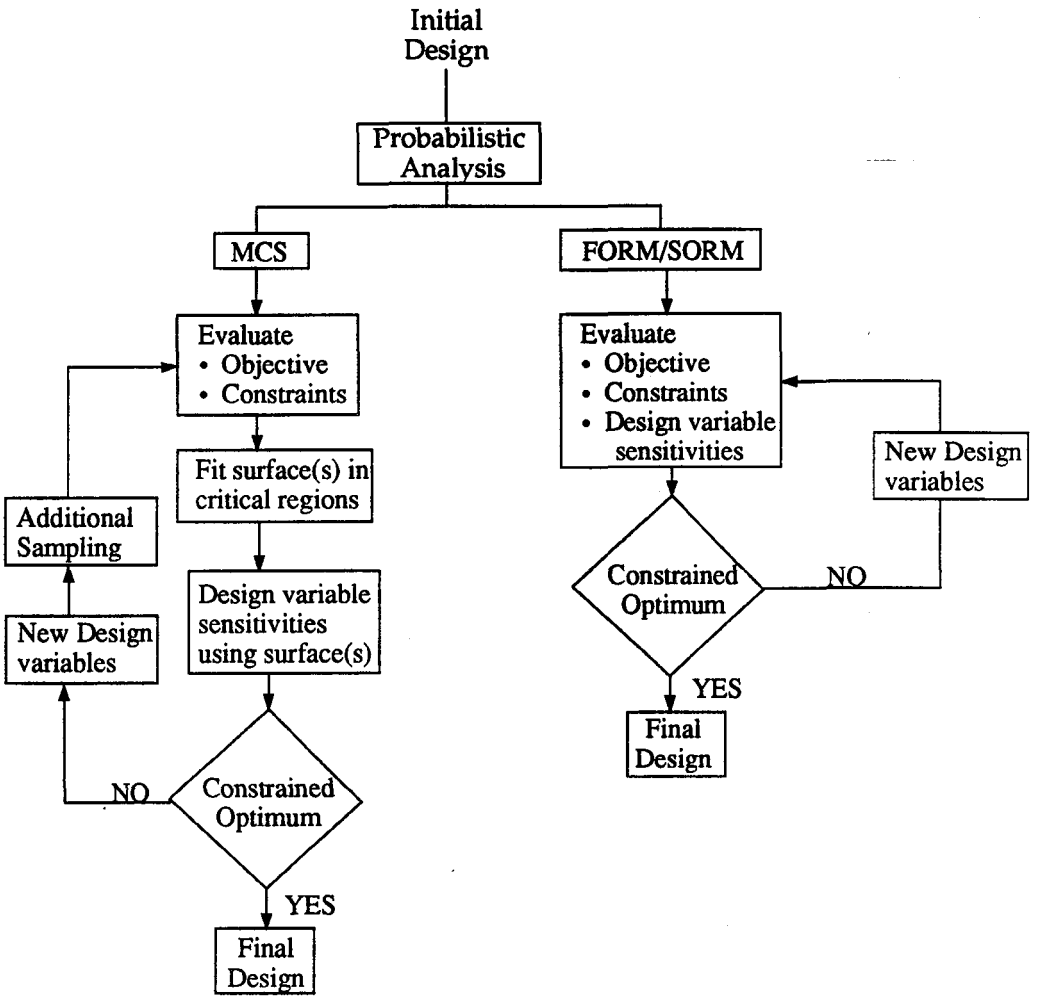

Figure 2-2. Solution Algorithm for Stochastic Optimization

Before describing the algorithm, it is important to recognize that the objective function and each of the individual constraint equations can be either deterministic, statistically stated or probabilistically stated. A statistically stated objective function is one wherein the objective is to minimize (or maximize) a statistical measure of response or performance. For example, the objective could be to minimize expected cost. A statistically stated constraint has a similar definition; for example, requiring that expected deflection be less than a specified value. A probabilistically stated objective or constraint is one that meets a probabilistic criterion. An example of a probabilistic constraint is a constraint that requires that the probability of exceeding ultimate stress be less than 99.99%. A probabilistically stated objective function would be, for example, to minimize probability of failure.

MCS Path. For the MCS path, the first step is to execute a simulation using the initial design configuration. From the simulation results we can directly and easily evaluate the objective function and constraints, whether deterministic, statistically stated or probabilistically stated. In MCS, the objective and constraint equations are repeatedly evaluated, each time with different realizations of the problem random variables. Each repeated evaluation is referred to as a single history or sample and is equivalent to a deterministic analysis. The realization of each random variable used for each history is obtained using a uniform random number generator and the inverse of the cumulative distribution function for the random variable.

The second step in the MCS path is to fit a response surface to the MCS results. The response surface provides a closed form equation to evaluate the value of the objective and the constraint equations given values of the input variables. Since each history in the MCS generates a response value for the objective and each of the constraint equations, we have generated a very rich source of data for response surface fitting. For constraint equations, it is important that the response surface be an accurate discriminator of whether or not the constraint is violated. For example, if the constraint specifies that maximum deflection be less than a constant, d, (with probability 95%) then the response surface must be most accurate in the region where values of the input variables cause the response to shift from less than d to greater than d. This is the so-called *critical* region. An additional criterion for multivariate problems is that within the critical region the fit should be most accurate where the values of the random variables are most likely to fall (these two criteria are actually analogous to the concept of most probable failure point (MPP) used in FORM/SORM structural reliability methods).

Response surface fitting proceeds using standard multivariate nonlinear regression analysis with weighting factors to ensure accurate fit in the critical region. The weighting factor applied to each data point is a normalized difference. For example, continuing with the deflection example described above, if a particular MCS history gives a deflection of x, then this sample point is weighted by $d/\sqrt{(d-x)^2}$[1]. Since the critical region will be different for each constraint, we use a different response surface fit for each constraint equation (continuing with the FORM analogy above, FORM analysis would have a different most probable failure point for each constraint or performance function). The second fitting criterion, that the fit should be most accurate in the space(s) within the critical region where the values of the random variables are most likely to fall, is automatically satisfied because the MCS will naturally generate more sample points in this space. Since more points are generated in this space, the regression procedure will force the surface to fit best in this space.

The response surface fits are now used to execute the optimization loop. On each iteration of the optimization loop, the values of the design variables are changed (as directed by the optimization algorithm) and the objective reevaluated to determine if the optimum has been achieved. The optimization algorithm requires sensitivity

---

[1]Other weighting criteria can also be used, if desired.

analyses and since the optimization is stochastic, each sensitivity is with respect to changes in mean values of the stochastic design variables. Therefore the sensitivities require MCS to be performed. However, these simulations use the response surface and are obtained with minimal computational effort since the response surface is analytic (and they can be executed in parallel).

Notice that the flow chart shows that the response surfaces are refit on each execution of the optimization loop. This is included because the location of the MPP changes on each iteration (because the mean values of the design variables change). Generally, it is not necessary to include this step unless there are large changes in the mean values and the system response is highly non-linear, since the original response surfaces remain accurate in the critical region. Refitting is performed, if necessary without any additional sampling. The refits are performed by reweighting the original sample points based on their distance to an estimate of the MPP. The additional sampling shown in the flow chart is only performed when there are large changes in the value of a design variable such that the space of the design variable was not adequately covered in the original simulations. Preliminary deterministic designs and small sample MCS stochastic designs can be performed prior to the full stochastic optimization to avoid this situation. In all cases, a complete stochastic analysis, encompassing a new simulation, or a FORM/SORM analysis should be performed at the end to validate the final design.

FORM/SORM Path. The FORM/SORM path is similar to stochastic optimization that has been used by others [Mahadevan 1992] and follows directly from the general formulation presented earlier. Hence, we provide only a brief discussion here. For this path, the probabilistic evaluations of the objective and constraint equations is by First or Second Order Reliability Methods. These methods require that we search for the most probable failure point (MPP). In finding the MPP it is necessary to perform sensitivity analyses, with respect to all of the problem random variables; however, once the MPP is found we do not need to perform additional sensitivity studies for the optimization loop [Mahadevan 1992]. The optimization algorithm updates the mean values of the problem random variables, using the available sensitivity information, and the probabilistic analysis is repeated. The entire loop is repeated until convergence to the optimum design is achieved.

We should point out that when using the FORM/SORM method, response gradients can be computed either by finite difference or by the direct differentiation method [Sues et al. 1985; Liu et al. 1987]. The direct differentiation method provides the greatest computational efficiency and accuracy, but at the expense of additional initial code development. Computational efficiency of the direct differentiation method over the gradient method can vary from a minimum of a factor of 2 to orders of magnitude, depending on the problem nonlinearity because solution for the gradient of a nonlinear response by direct differentiation requires the solution of linear equations (rather than nonlinear equations) on the order of the original system [Sues et al. 1985; Liu and DerKiureghian 1991].

Computational Effort. The computational effort is a function of the number of random variables, design variables, constraints, and constraint reliability goals. However, which path will execute most quickly must also consider the degree of parallelism and system hardware. We present next a quantitative assessment of the computational effort for each solution path for a sequential computer for the particular problem. Following this we discuss parallelization.

The computational effort for the FORM/SORM path on a sequential computer is on the order of:

*NCE=M*NSOL*(Number_Design-Iterations)*

where

NSOL=(Number_Random_Variables)(Number_Constraints+1)*

and *M* is typically between 3 and 10.

The computational effort for the Monte-Carlo simulation path (MCS) is on the order of :

*NCE=NSOL*S*

where

NSOL=10/(1.0-max[R])

and *max[R]* is the maximum reliability goal of any constraint equation and *S* is proportional to the additional sampling required when large changes in design variables occur during the design iteration and the response surface is updated. Typical values of S will be between 2 and 3 (allowing for a full simulation at the end of the design for validation).

Each path is also highly parallel. For the MCS the majority of the computational effort is in the initial simulation. However, massive parallelism can be achieved with high efficiency for MCS as demonstrated in earlier research [Sues et al. 1993; Sues et al. 1992]. Hence, on a massively parallel machine, MCS will likely be the selected path. For the FORM path we can also achieve a high degree of parallelism; first by evaluating sensitivity coefficients for the random variables in parallel. Ability to achieve extremely high parallel efficiency for sensitivity analysis is demonstrated for this research and is described in the next chapter. In addition each constraint equation will have a different most probable failure point (MPP); hence evaluation of each probabilistic constraint requires an independent FORM (or SORM) analysis that will be executed in parallel. For problems wherein FORM is computationally less expensive (as given by equations above) and it is possible to effectively use all available processors (that is, achieve a high enough degree of parallelism) then the FORM path would be selected.

2.4 PARALLELISM IN MULTI-DISCIPLINARY STOCHASTIC OPTIMIZATION

2.4.1 Sources of Parallelism in Coupled Aeromechanical Design

There are six general classes of parallelism in stochastic optimization problems for coupled aeromechanical design that we have identified: (1) parallelism in the general probabilistic computations; (2) parallelism in optimization algorithms; (3) specialized parallelism that arises in stochastic optimization; (4) parallelism in the general structural mechanics computations; (5) parallelism in the aerodynamics computations; and (6) parallelism inherent in multi-disciplinary analyses. Sources of parallelism in items 1, 2, and/or 6 are coarse grained while the additional levels of parallelism dealing with items 3, 4, and 5 can be coarse to fine-grained.

General Probabilistic Computations. Methods for parallelizing the general probabilistic computations (Item 1) have been reported previously [Sues et al. 1991a, b] and are summarized in Table 2-2. These sources of parallelism are very coarse-grained and very high parallel efficiencies can be achieved. In addition a very high degree of parallelism can be achieved for the Monte-Carlo Simulation method (of course, this is also often, but not always, the most computationally intensive method).

Optimization. In optimization (Item 2) parallelism arises in the search to find the optimum design solution. Optimization algorithms can use any one of a number of search strategies to find an optimal solution; however, all require repeated evaluation of the objective and constraint functions for different trial values of the design variables. Most commonly, search strategies involve computation of response gradients with respect to each of the design variables in order to update the design solution. These response gradients or sensitivity coefficients can each be computed independently and in parallel. This source of parallelism is very coarse grained and nearly perfect linear speedup can be achieved as will be demonstrated in the next chapter. The main limitation of this strategy is that the degree of parallelism is limited to twice the number of design variables (or just the number of variables if one-side finite differences are used to estimate the response gradient).

Stochastic Optimization. For stochastic optimization (Item 3), sources of parallelism in the two-path approach (Figure 2-2), were introduced in the preceding section. For stochastic optimization the sources of parallelism derive primarily from the stochastics since the optimization sensitivities are obtained as a by-product of the probabilistic analysis. As pointed out earlier, the MCS path has the same degree of parallelism as in a general probabilistic analysis and high parallel efficiency can be achieved as reported in previous research [Sues et al. 1993; 1991a, b]. The FORM path, however, can also have a high degree of parallelism since it incorporates two of the sources of parallelism shown in Table 2-2: (1) repeated performance function evaluations for perturbed inputs; and (2) multiple failure mode analysis. The first source of parallelism is in the computation of response gradients with respect to the problem random variables that arise in FORM. The second source of parallelism arises from the multiple reliability-based constraints in the stochastic optimization. Each reliability-based constraint will have a different design point (MPP) and, therefore,

requires an independent FORM analysis at each design iteration. Hence, while the degree of parallelism will not be as high for the FORM path as the MCS path, it can be large enough to effectively use tens of processors for many typical problems.

TABLE 2-2. SOURCES OF PARALLELISM IN PROBABILISTIC MECHANICS

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.

General Structural Mechanics. Parallelism in the general structural mechanics computation (Item 4) 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. In general, parallelism is exploited by using either domain decomposition methods, or substructuring methods or through the use of parallel equation solvers. For more details the reader is referred to the cited references.

Aerodynamics. Parallelism in aerodynamics (Item 5) arises from several sources. In the implementation studies presented in the next Chapter we exploit, quite successfully, parallelism in computation of aerodynamic influence coefficients. This parallel decomposition falls naturally from the nature of the linear potential analysis method used. In this method, the influence coefficients are mutually independent functions of the surface geometry. Therefore, they are inherently parallel.

Additional sources of parallelism in other aerodynamics methods have been identified and will be pursued in future research. For unsteady, compressible potential aerodynamics (for example, for propfan flutter analysis), the same source of parallelism exists as just discussed. For unsteady pseudo-three-dimensional cascade aerodynamics analysis (for general analysis of blades in a turbomachine compressor stage), parallel decomposition is achieved by performing requisite 2D cascade analyses in parallel. An additional level of parallelism can be obtained by analyzing multiple compressor stages each on separate clusters of nodes with flows between stages being modeled approximately. Finally, for 3D Euler analyses of a single blade row in a compressor stage (for detailed surface flow analysis), parallelism can be achieved by generating

multiple grids for several blade rows, and solving the separate blade rows in parallel. Similar approaches have been demonstrated in previous research by others [Blech et al. 1991]. Additional levels of parallelism can be obtained by using a parallel equation solver.

Multi-Disciplinary Design. Multi-disciplinary design problems have several unique sources of coarse-grained parallelism that require further investigation. The potential for parallel analysis arises from the concept of simultaneously executing analyses from the individual disciplines. Straightforward implementation is somewhat hampered when the disciplines are truly coupled (for example when structural response significantly alters the loading as in many fluid-structure interaction applications) and approximations based on iterative solution schemes are required to invoke parallelism. For aeromechanical analysis a source of multi-disciplinary parallelism that we plan to investigate in future research is simultaneous execution of steady and oscillatory calculations for both aerodynamics and structures. The unsteady analysis requires first that an oscillatory pressure distribution be known for the first 3 or 4 blade vibration modes. The 3 or 4 oscillatory pressure distributions can be calculated in parallel. The distributions feed into an additional parallel structural analysis. Once all calculations are complete, a master process combines the results to determine flutter constraint values. Similar parallel calculations can be performed for analysis and design of both propfan blades and compressor stage blades.

2.4.2 Computational Strategy for Multi-level Parallelism

As identified above there are many sources of parallelism in multi-disciplinary stochastic optimization problems. In general it is necessary to invoke more than one source of parallelism for one of two reasons: (1) increase the degree of parallelism; or (2) reduce memory/processor demand. For typical design optimization problems or probabilistic analyses by methods other than MCS, if only one source of parallelism is used, it will generally only be possible to effectively use tens of processors. Hence, additional sources of parallelism must be exploited to achieve the high parallel speedups we are after. Also, for problems with large memory requirements typical of many MSO applications, it is necessary to distribute computations over several processors to reduce memory demand per node (or use computational algorithms that minimize memory requirements). As a simple example, if 96 Mbytes of storage are required to solve a structure and only 16 Mbytes are available at each processor node, 6 processors at a minimum must be assigned to solve a single structure. Decomposition among these 6 processors must then be accomplished. In this example two sources of parallelism can be exploited simultaneously in a two-level decomposition to increase the degree of parallelism and manage the memory/processor demand. That is, for stochastic optimization we use clusters of 6 processors each to perform independent sensitivity analyses or Monte-Carlo simulation histories.

As indicated above by the simple example, the sources of parallelism can be arranged in a hierarchical structure with the coarsest grained parallelism at the top and successively finer-grained sources of parallelism at lower levels. By exploiting the

higher levels first we maximize the average granularity and thereby achieve the highest degree of parallelism with the coarsest possible granularity. The multi-level parallel decomposition strategy is depicted in Figure 2-3. In the next chapter, we implement and evaluate a two-level decomposition. In general, there are limits to the number of levels of parallelism that can be efficiently exploited. These limitations will depend on the communications bandwidth of the particular hardware platform and the compute/communicate ratios at each level of parallelism for the particular problem.

Level I. The top level decomposition is referred to as task farming and exploits the data-independent parallelism that is inherent in MSO problems (Section 2.4.1, generally items, 1, 2, and 3). Each data-independent computation, for example, individual history computations for the MCS path or the response sensitivity computations for each probabilistic constraint evaluation for the FORM path is referred to as a task.

Level II. The second level decomposition focuses on data decomposition of the structure into sub-domains or matrix partitioning depending on the solution method used (Section 2.4.1, generally items 4 and 5). The number of decompositions to use (that is the number of sub-domains or matrix partitions) must consider that while increasing the number of decompositions allows more processors to be used, parallel overhead will increase. In fact, negative speedup can result if too many decompositions are used. Also, the decompositions must be selected so that load balance among the processors can be achieved. As an illustration of two-level decomposition consider a case where a structure is broken into four domains. Here four processors would work together as a cluster on each of the Level 1 tasks. Similarly if the Level II decomposition of the aerodynamics is such that the flow field is broken down into four portions (e.g., four compressor stages) then four processors would work together as a cluster on each of the Level 1 tasks.

As will be demonstrated in Chapter 3, Level I and Level II decomposition can be highly efficient on massively parallel processors. For networks of workstations, degradation can occur when Level II is invoked, if the granularity of the Level II computation is fine. However, it will not generally be necessary to invoke large numbers of Level II subdomains on workstation networks, since workstation networks will have limited numbers of nodes and also, very large amounts of memory per node.

Level III. The third level decomposition that can be invoked, if necessary, is at the element level. This level of parallelism is a fine-to-medium grain data decomposition. At this level the structure subdomains can be further decomposed and additional multiple processors used, for example, to perform constitutive model evaluations for different elements, including micro-mechanics analyses for composite materials. The need for Level III data decomposition will be evaluated in future research; and will depend on specific commercial applications/needs and advances in hardware (that is, to even larger numbers of processors that in currently emerging hardware).

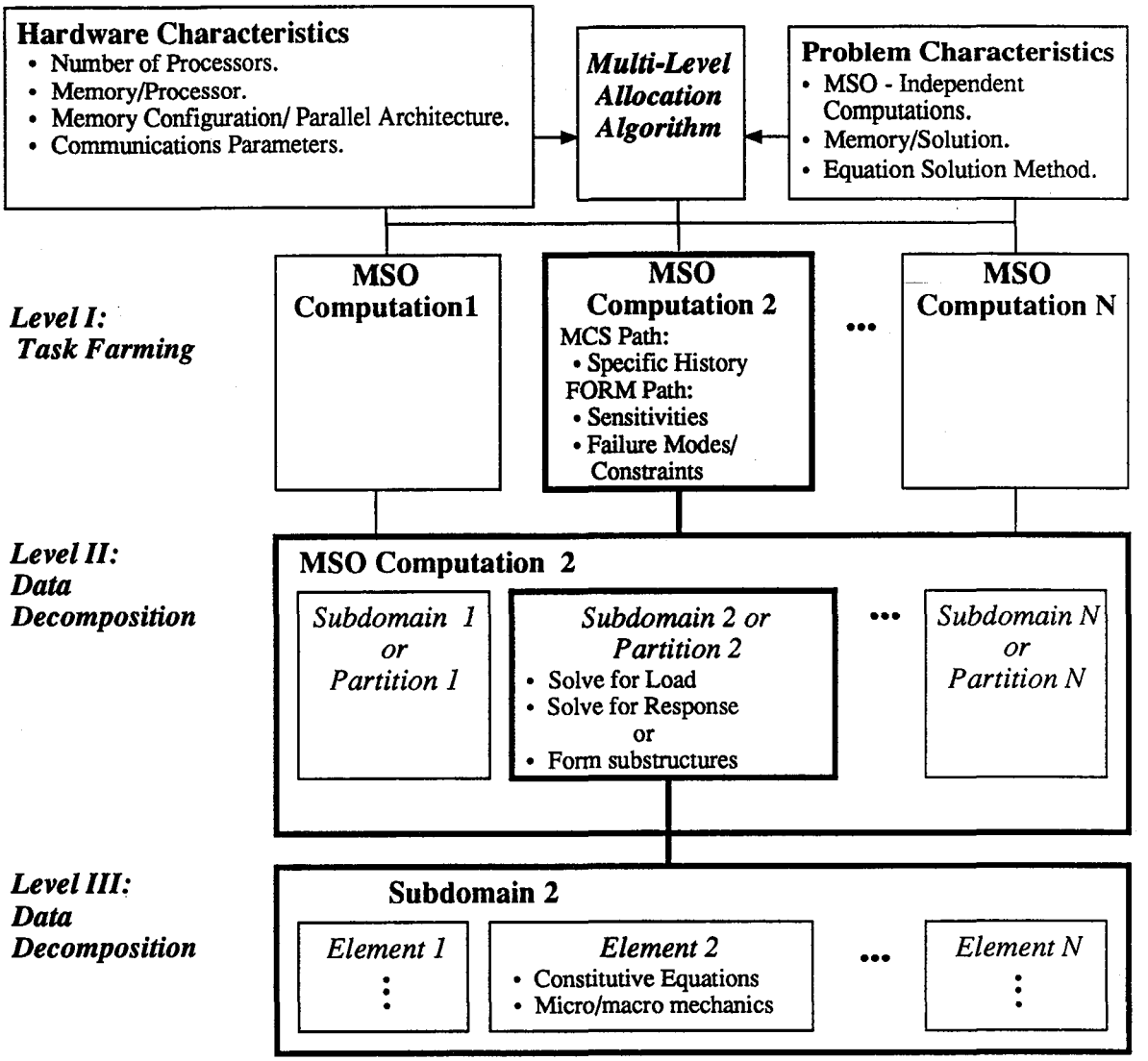

Figure 2-3. Top-Down Strategy for Parallel MSO Problem Decomposition

Parallel Control Algorithm. Phase II of this research will develop a parallel control algorithm that uses the hardware and problem characteristics to determine the optimal multi-level decomposition. The pertinent hardware characteristics include: (1) the number of processors; (2) the architecture type (shared-memory, distributed-memory, etc.); (3) the amount of memory per processor; and (4) communications parameters (network configuration, data transfer rates/bandwidth, etc.). The pertinent problem characteristics include: (1) the number of independent problem solutions required by the stochastic optimization (NSOL—see Equations 2-1 and 2-2); (2) the amount of memory required for each solution; and (3) the equation solver selected. Details of the parallel control algorithm are presented in our Phase II research proposal.

2.4.3 Special-Purpose Computational Algorithms

In addition to the multi-level parallelization strategy, a key component of this research is the identification of special purpose algorithms that are not only parallelizable but reduce the basic computational effort in performing the repeated objective function and constraint equation evaluations that are required for the probabilistic analysis and optimization and at the same time minimize memory requirements (a key to achieving high parallel efficiency). Two approaches have been identified and are briefly summarized below, probabilistic substructuring and the SPCG equation solver.

Probabilistic Substructuring and Substructured Optimization. Probabilistic substructuring can be used prior to execution of the probabilistic computations in order to reduce the memory/processor requirements and to reduce the execution time of each structural response solution required by the probabilistic analysis. Probabilistic substructuring has been successfully demonstrated in earlier research for solving fatigue reliability problems [Sues et al. 1993].

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

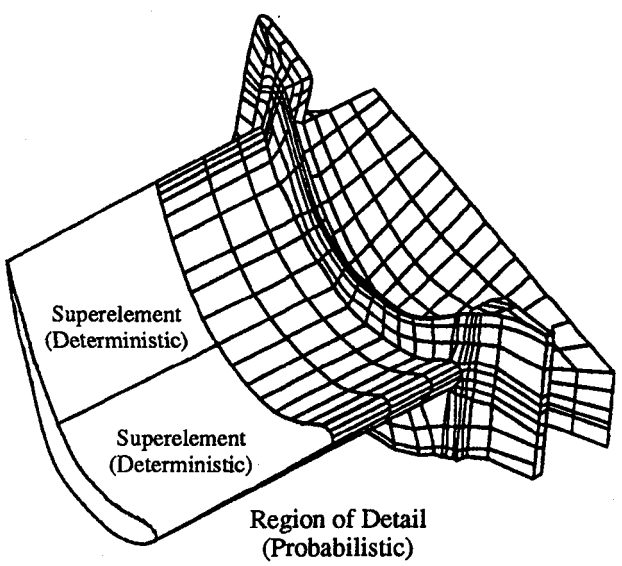

Figure 2-4. Probabilistic Substructuring

For parallel implementation, multiple processors are first used to develop each of the super elements. This is accomplished by assigning one processor per super element. Once the super elements 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 super element are treated deterministically; however loadings on the super element can still be treated as random. The key contribution of this approach, however is in greatly reducing the memory requirements of the probabilistic analysis which is known to have a great impact on the parallel efficiency.

The probabilistic substructuring approach can also be applied for parallel deterministic optimization. For deterministic optimization, savings are achieved because the design sensitivities can be computed at a much reduced cost and memory requirements are reduced. Of course properties within the super element and at the super element boundary cannot be design variables. However, outside the super elements in the regions of detail, size and shape are free to vary to obtain an optimum structure.

Stochastic Preconditioned Conjugate Gradient Solver. The stochastic preconditioned conjugate gradient solver is a very effective procedure for solving large systems of equations in probabilistic finite element analysis [Sues et al. 1992]. Computational efficiency is achieved through the use of a pre-conditioning matrix that can reduce the number of iterations required for convergence in the conjugate gradient scheme by an order of magnitude. The procedure obtains the pre-conditioning matrix from the central solution obtained at the start of every probabilistic analysis (the solution obtained using mean values of all the random variables) at essentially no added cost. While the procedure was developed for application in probabilistic analysis, it is a general purpose approach for sensitivity analysis, and hence is identically applicable for deterministic optimization. Although the above-referenced study implements the solver sequentially, it is ideally suited for parallel implementation since it is an iterative approach requiring only matrix-vector multiplications and vector dot products. The solver has also been shown to be efficient for use with sparse storage schemes so that memory requirements can be minimized for large 3-D problems.

CHAPTER 3

PARALLEL IMPLEMENTATION USING PARALLEL VIRTUAL MACHINE (PVM)

3.1 INTRODUCTION

Hundreds of different kinds of computer systems exist today. These systems vary significantly in type of processor, operating system, peripherals, connectivity, etc. In order to provide an application which is useful to most of the people using computers, a software developer must somehow support multiple systems. In many cases, particularly for engineering analysis software and packages which do not produce graphical output, the problem of supporting multiple computer systems is reduced to supporting multiple operating systems. The most widely used operating systems in the world are MS-DOS and PC-DOS, Apple Macintosh, and Unix. For powerful engineering analysis and design applications, Unix is the most popular system. However, every engineering workstation manufacturer publishes its own version of Unix, each with added features and modifications. Many vary significantly from the AT&T and Berkeley standards. On top of the operating system differences, different systems will likely contain compilers and development tools with differences that may exceed the differences between operating systems.

While it is clearly difficult to produce one source code for a major application which is portable and will compile and run on every version of Unix, it is even more difficult to produce a portable parallel application. The differences in operating systems, compilers and tools often manifest themselves in networking services and inter-application communication. This is especially true of multiple-processor parallel computer systems.

Our approach toward developing a portable parallel multi-disciplinary stochastic optimization (MSO) computing environment is described in this chapter. We also provide the results of several parallel implementation and timing studies, including the multi-level decomposition approach described in the previous chapter.

3.1.1 ANSI Standard Programming Languages

Every version of Unix provides two compilers, FORTRAN and C, which are fairly portable. In fact, although suppliers generally provide language extensions, they usually support the ANSI language standards. For this reason, it is important that code intended to be portable be written strictly in an ANSI standard language. There is no guarantee that multiple compiler vendors support any language extensions, and code which uses extensions will likely encounter problems. The demonstration problem code was developed in the ANSI standard FORTRAN programming language, using extensions only to obtain timing data.

3.1.2 Portable PVM

Networking and inter-application functions are not currently defined within the ANSI FORTRAN and C standards. Certain key functions are common to all versions of Unix; however there are some variations. Also, parallel implementations of Unix have additional added functions for inter-processor communication, and these functions vary from manufacturer to manufacturer. For these reasons, it is impossible to write a portable parallel source code which uses native system calls. The demonstration problem code was developed using Parallel Virtual Machine (PVM) version 3.1, created at Oak Ridge National Laboratory described in Chapter 2.

3.2 PARALLEL COMPUTER SYSTEMS

In order to demonstrate that the PVM approach to MSO executes on a wide variety of computer systems without modification to the parallel function calls in the source code, the demonstration problem was executed on the following systems:

1. ***Intel iPSC/860 Hypercube.*** The Intel iPSC/860 installed at NAS at NASA-Ames is a Multiple Instruction, Multiple Data (MIMD) distributed memory parallel supercomputer. It has 128 Intel i860 nodes operating at 40Mhz connected in a hypercube architecture (Figure 3-1). Communication between nodes is by way of direct-connect routing supporting variable length messages and peak data rates of 2.8MB per second on 8 bi-directional connections at each node. Each node contains 8 MB of RAM, for a total of 1GB of memory. The hypercube is connected to a Concurrent File System which is an array of 10 high-speed SCSI disks, providing 7MB of mass storage. The iPSC/860 is connected to the NAS network through a single computer system known as the System Resource Manager (SRM), which runs a version of the Unix operating system. The iPSC/860 version of PVM consists of two components. The host component runs on a Silicon Graphics workstation located on the NAS network. The workstation is responsible for allocating hypercube resources by issuing network requests to the SRM. All external communication to the hypercube is routed to the SRM. The node component of PVM actually runs on each hypercube node.

2. ***Lewis Advanced Computing Environment (LACE) Cluster.*** The NASA-Lewis LACE Cluster is a network of 32 IBM RS/6000 560 workstations, all of which run IBM's AIX version of Unix. The workstations include the standard array of Unix compilers. The cluster is divided into two subnetworks. Eight of the workstations are allocated for batch and interactive processing. Of these, one has 512MB of RAM memory, and seven have 128MB of RAM. The remaining 24 workstations are dedicated to MPP and parallel processing. The MPP nodes each have 64MB of RAM. Through special procedures, it is possible to confine MPP work to the MPP nodes so that parallel processing applications will not have to compete with additional network traffic between the two subnets.

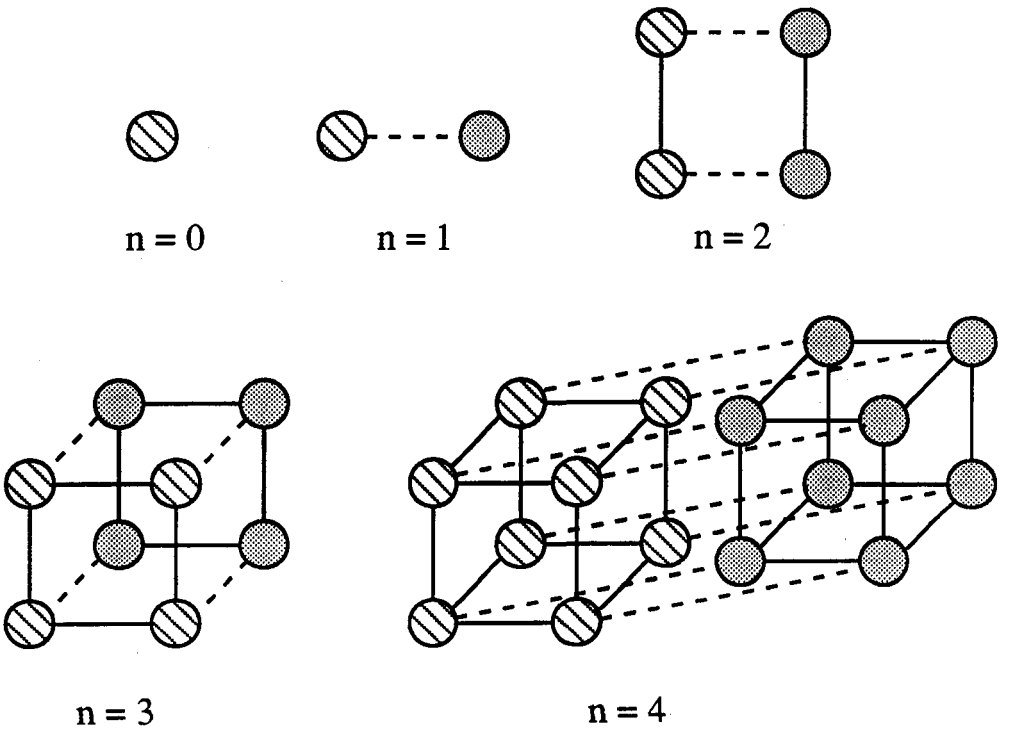

Figure 3-1. Hypercube Architecture. An (n+1)-dimensional cube is constructed by replicating an n-Cube and then connecting each vertex in the original cube with its corresponding vertex in the replicated cube. Number of nodes is 2^n; number of connections per node is n; max distance between nodes is n.

3. ***Hewlett-Packard 9000 Series 730 Workstation.*** Applied Research Associate's HP730 single-processor workstation was the primary development platform for the demonstration code. The HP contains 32MB of random-access memory, and 1.8GB of disk space, and runs Hewlett-Packard's HP-UX version of Unix. The demonstration code was tested on the single-processor HP, with multiple processes representing multiple nodes.

3.3 ADVANCED PROPFAN BLADE DEMONSTRATION PROBLEM

3.3.1 Problem Description

The NASA/Industry Advanced Turboprop Project (ATP) was begun in 1978 to investigate the possibilities of using fuel efficient turboprops in the place of current turbojets and turbofans [Ziemianski, 1988; Hager, 1988; Rhodes 1991]. Current straight-bladed turboprop propeller blades are fuel efficient at Mach numbers below approximately 0.6, but lose this efficiency rapidly at higher Mach numbers largely due to compressibility effects. The ATP project included a major effort to design new propeller blades, known as propfan blades, which operate efficiently at the higher cruise Mach numbers (around 0.8) of turbojet and turbofan aircraft. These advanced propellers include highly swept blade planforms, thin sections, and a large number of

blades per engine for both single-rotating and counter-rotating configurations. These propellers have an installed propulsive efficiency which is increased by 10 to 20% from turbofans (Figure 3-2). Flight tests have shown that this increased efficiency from the new propellers alone could provide a fuel savings as large as 30%. A combination of improved propeller design and modern turbine engine technology offer a potential fuel savings of 50% over equivalent-technology turbofans.

The example problem for this Phase I research focuses on the aerodynamic design of an advanced propfan blade. This problem contains many of the salient features of typical aeropropulsions system design problems. The real-world design of such blades must consider many different engineering disciplines. A complete design may include hundreds of design parameters and constraints. For example, the blade must operate at cruise Mach numbers of approximately 0.8. In order to maximize propulsive efficiency, the blade must have minimal supersonic flow on the surface in order to reduce losses due to shock waves, and so the blade leading edge must be swept back. Supersonic flow is further reduced by designing very thin blade sections which exhibit only weak shocks near the trailing edge. The blade will be highly loaded and so the structural design of the blade is important, particularly with the thin, swept sections. In the presence of a fuselage/wing/nacelle configuration, there is an unsteady, periodic loading on the blades which causes the blades to load and unload with every revolution. Additionally, there is potential for shock waves on the blades which may themselves oscillate, enhancing the unsteady loading. A further source of vibration and

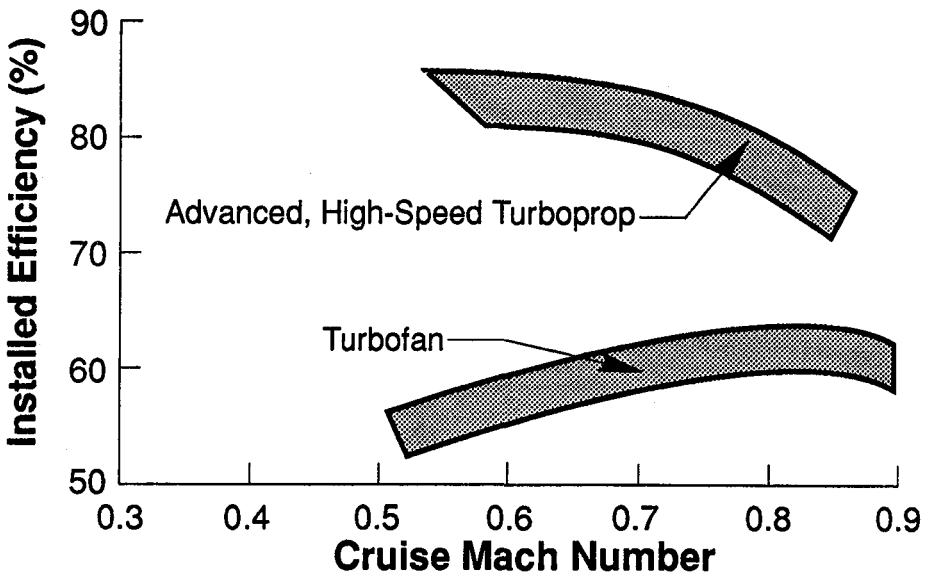

Figure 3-2. Increased Propulsive Efficiency with High-Speed Turboprops

unsteady loading is stall flutter due to flow separation and periodic loadings at very low Mach numbers, and classic flutter at high Mach numbers. These unsteady loadings present an additional structural burden, and may reduce the life of the blade due to fatigue. A third discipline involved is aeroacoustics. The blade oscillation will create a noise element which often exceeds the noise due to the engine. For example, during takeoff, the blade rotation rate will be high, and the community noise pollution created during takeoff can present a severe problem near airports. The solution to this problem is found by optimizing propulsion system and airframe integration such that the propeller noise is reflected away from the community. Thus, for the design of a propfan blade, engineers must consider, at a minimum, aerodynamics, structural mechanics, and aeroacoustics to create an acceptable, real-world design. The design which is technically the best may not be feasible due to operating costs, and so operating cost must be considered to create a design which is practical.

Since propeller technology alone contributes the majority of the fuel savings in an advanced turboprop aircraft, it is desirable to extract as much of this potential as possible. Historically, the design of propellers has been quite separated from the efficient design of aircraft. Many of the propellers in use today were designed using the methods of Goldstein and Theodorsen [Goldstein, 1929; Theodorsen, 1948] purely to analyze some chosen twist and chord distribution or to produce a chord and twist distribution based on an optimal radial loading. These methods did not consider sweep; however, much experimental propeller research was performed on swept-bladed propellers in the 1950's. This research indicated swept propeller blades could perform efficiently to Mach numbers of approximately 0.85. However, it was found that these blades tended to flutter and break. From the mid-1950's until the ATP project was started, propeller research was at a standstill. During the ATP project, many parametric studies were performed to determine the best aerodynamic shape for an advanced propfan blade. These studies borrowed blade planform concepts from the experimental research in the 50's. The ATP research included analytic and digital computer analyses as well as experiment. The primary research effort of the ATP project focused on a set of perhaps 10-15 different blade configurations. The configurations varied in number of blades per engine, blade sweep and twist, and chord distribution. Additionally, both single-rotating and counter-rotating configurations were considered. Single-rotating propellers have an induced swirl in the flow behind the propeller, and this swirl is an indication of a loss of propeller efficiency. It was supposed that a second set of blades behind the first, rotating in the opposite direction, would remove the swirl from the flow and increase the propeller efficiency. The ATP project included a wide range of analyses, including aerodynamic, aeroelastic and structural, and acoustic. The analyses were both computational and experimental. The experimental tests were performed on small, dynamically scaled models, such as the one in Figure 3-3, and on full-size aircraft. However, the entire ATP project was based on a select few propeller configurations, and the blade sections which were determined to be appropriate were only the best from the configurations available. A well-defined MSO computer application could

have produced a propeller which performs better than the best propeller from the ATP project.

The geometric shape of a propfan blade is defined primarily by radial distributions of chord length, 2D airfoil section, leading edge sweep, and twist angle. The twist angle, β, is measured from the plane of propeller rotation to the chord of the airfoil at a given radial station. Typically, the primary propeller performance parameters, thrust coefficient C_T, power coefficient C_P, and propeller efficiency, η, are plotted versus the non-dimensional *Advance Ratio, J*, which is defined by:

$$J = \frac{V}{nD} \tag{3-1}$$

where V is the free stream velocity, D is the blade diameter, and n is the blade rotational speed in *revolutions* per unit time. The non-dimensional coefficients and the propeller efficiency are defined as:

$$C_T = \frac{Thrust}{\rho n^2 D^4} \tag{3-2}$$

$$C_P = \frac{Power}{\rho n^3 D^5} \tag{3-3}$$

$$\eta = \frac{C_T}{C_P} J \tag{3-4}$$

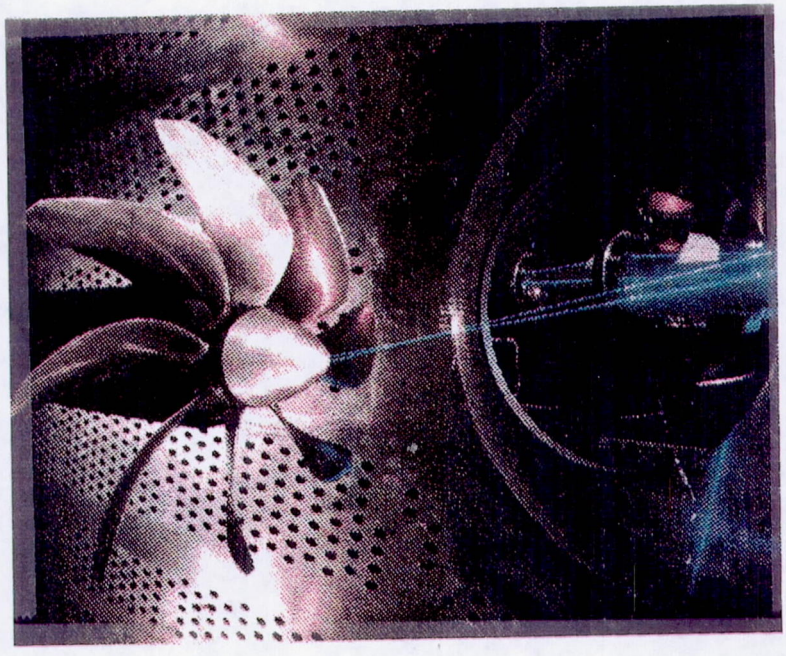

Figure 3-3. Typical Advanced Propfan Blade Experimental Model

The power is the theoretical shaft power required to drive the propeller at a given J, and can be related to aerodynamic torque, C_Q, by:

$$C_p = 2\pi B C_Q \tag{3-5}$$

where B is the number of blades and torque coefficient, C_Q is:

$$C_Q = \frac{Torque}{\rho n^2 D^5} \tag{3-6}$$

In order for the propeller to operate at a specified J, the engine must provide a torque, C_Q, as defined above. The propeller will absorb the corresponding C_P, as given in Eq. 3-5. It is likely that, for a particular classification of aircraft, only a few candidate engines will be available. The fuel consumption rate for the engine will be known at various throttle positions, as will the available power and torque. In designing a propeller for single-point operation (*e.g.* at cruise), some candidate fuel consumption rate will be selected, and this will define a design shaft power for a particular engine. The propeller must be designed with the constraint that the required power be equal to the available power at the design J.

For the example problem, the selected objective is to maximize the thrust C_T provided by an 8-bladed, single-rotating SR-7 propfan blade [Aljabri 1987] for an Advanced Turboprop aircraft in a takeoff/climb-out configuration. The flight Mach number is 0.165 which corresponds to a sea-level airspeed of 125 mph, and a power input of 2730 shaft horse-power (C_P=.94) at 1350rpm. The blade to be designed has a diameter of 9 feet. This low Mach number case was selected to confine the problem scope to subsonic flow in the Phase I research (this simplifies the analysis but still allows us to investigate parallelization issues). From the given information, a design advance ratio J=0.9 is calculated, and this is the operating condition variable for the blade. For the example, the airfoil and chord are specified at 10 discrete positions down the span of the blade. The 10 design variables to be selected by the optimization algorithm are the blade twist angles. In practice, the sweep would also be an additional series of design variables, and flutter analysis would be included. Two different optimizations were performed: one to optimize the blade shape at the power coefficient of the SR-7 blade at the design advance ratio and one to optimize the blade shape for a power coefficient ten percent greater than the SR-7 power coefficient. The initial shape in each case was the SR-7 shape.

3.3.2 Brief Description of the VORP Aerodynamics Code

The propfan blade performance analysis in the demonstration code is performed using the ARA VORP code. VORP is a linear-potential panel method which solves flows for which there is exclusively incompressible flow on the blade surface. VORP uses the two-dimensional Prandtl-Glauert correction at each radial blade station to approximate the compressible subsonic flow, as described below. The demonstration optimization is

performed for free-stream Mach numbers for which the blade tip Mach number is significantly less than unity in order to guarantee there is no supersonic flow (which is not modeled by VORP) on the blade. A summary of the aerodynamic method used by the VORP code is given below.

Mathematically, the motion of a fluid can be described using either of two methods. The *Lagrangian* method models the motion of discrete mass elements of the fluid. Although the Lagrangian method treats discrete masses, it does not in general treat the motion of individual molecules. The *Eulerian* method focuses on the fluid properties at various points in space, and the variation of these properties with time. The VORP code is based on the Eulerian methodology, under the assumptions that the flow is inviscid and irrotational. A fluid flow will always behave under the laws of continuity (conservation of mass), conservation of linear momentum, and conservation of energy. If it is assumed that viscous layers are thin and negligible, the continuity equation can be written as:

$$\frac{\partial \rho}{\partial t}+\nabla\cdot\left(\rho\vec{V}\right)=0 \tag{3-7}$$

where ρ is the fluid density and $\vec{V}$ is the fluid velocity. If it is further assumed that the fluid is steady and incompressible, the continuity equation becomes:

$$\nabla\cdot\vec{V}=0 \tag{3-8}$$

In a steady flow for which there is no boundary layer separation, it is reasonable to assume that the flow is irrotational. This is sufficient to introduce a velocity potential function, Φ, such that:

$$\vec{V}=\nabla\Phi \tag{3-9}$$

Then the continuity equation becomes Laplace's equation,

$$\nabla^2\Phi=0 \tag{3-10}$$

Such an inviscid, irrotational flow is known as a potential flow, and Laplace's equation for the velocity potential is the governing partial differential equation for such flows. Laplace's equation is an elliptic, linear partial differential equation to which the principle of superposition may be applied. If multiple solutions are known, then any linear combination of the solutions is also a solution. This characteristic simplifies the calculation of the inviscid, incompressible, irrotational fluid flows.

The solution to the potential flow problem is the velocity potential field throughout the volume in which the flow is contained. The velocity at every point is calculated directly as the gradient of the potential. Through the use of Green's identity in the second form, the problem reduces to a boundary-value problem. For a flow

problem, there are two boundary conditions. The first condition is that the flow is normal to every solid boundary. The second boundary condition is that the perturbation in velocity potential due to solid bodies being present must diminish to zero at an infinite distance from the body in any direction. The VORP code is a panel method which solves for the velocity potential field by selecting a distribution of singular, elementary flows over the surface of all solid bodies; in this case, the propfan blades, such that the boundary conditions are satisfied. At any point in the flow field, the total velocity potential can be found by integrating the perturbation potential at the point due to the elementary flows on the solid bodies and adding the free-stream potential. The flow singularities are quadrilateral vortex rings which lie in a lattice grid on the surface of the propfan blade, as shown in Figure 3-4. The velocity potential of a potential vortex satisfies Laplace's equation, and the velocity perturbation due to a potential vortex diminishes to zero at infinity, automatically satisfying the second boundary condition. The helical wake attaches to the trailing edge vortex rings. This wake is necessary in an irrotational flow in order for a lifting force to be generated. VORP uses the 2D Kutta condition at each spanwise blade station as an approximation of the wake boundary condition. In the figure, the influence coefficient is defined as the normal component of the velocity perturbation at an element's control point due to a vortex ring of unit strength on the influencing panel. The perturbation velocity is calculated using the law of Biot and Savart and is described in detail by Bertin and Smith [Bertin and Smith, 1979]. The influences are dependent only on the geometry of the problem and can be calculated in parallel. The system of equations becomes:

$$\sum_{j=1}^{n} C_{ij}\Gamma_j = -\left(\vec{V}_\infty + \vec{V}_{i,\,rotation}\right)\cdot\hat{n}_i,\ \text{i=1 to N} \qquad (3\text{-}11)$$

where $\vec{V}_\infty$ is the free stream velocity, $\vec{V}_{i,rotation}$ is the velocity due to rotation at the control point of element i, and $\hat{n}_i$ is the outward unit normal vector of element i at the control point. Once the vector solution for Γ's is known, a 3D equivalent to the Kutta-Joukowski theorem, which relates force to vortex strength, can be used to determine the aerodynamic force generated on each element [Karamcheti, 1966]. The integration of the forces yields a thrust (axial force) and a torque (moment about the axis of rotation), which are reduced according to equations 3-2 and 3-6, respectively. Once C_T and C_Q are known, C_P and η can be determined for the problem. VORP accounts approximately for compressibility effects by using the 2D Prandtl-Glauert correction [Bertin and Smith, 1979] at each spanwise blade row.

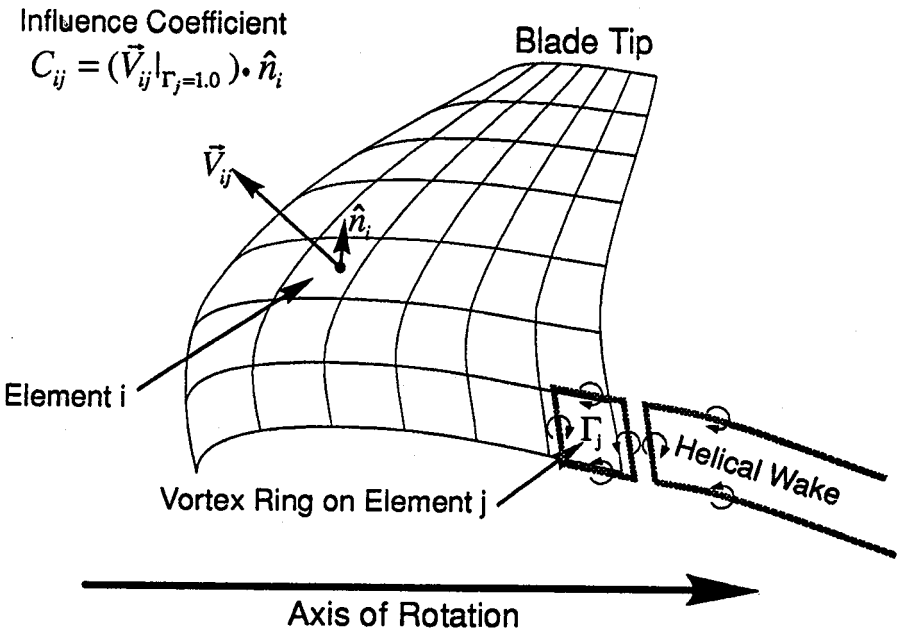

Figure 3-4. Representation of the Flow Field in the VORP Code

$$C_{Force,compressible} = \frac{C_{Force,incompressible}}{\sqrt{1-M_\infty^2}} \tag{3-12}$$

where in the case of a propfan blade, M_∞ is the Mach number due to both the free-stream velocity and rotation.

3.4 IMPLEMENTATION OF THE MSO

3.4.1 Overview

The software developed and implemented in Phase I does not represent the complete MSO methodology. Rather, it represents key features of MSO, in order to demonstrate the feasibility of further development. The coding was performed using the Parallel-Virtual-Machine (PVM) version 3.1 toolkit described in Section 2.2.

For optimization, we use the Automated-Design-Synthesis (ADS) optimization library, which is a NASA code supported through COSMIC. This library is flexible and proven and contains several optimization schemes which work well for unconstrained and constrained optimizations problems with from one to approximately 40-50 design variables. The ADS code is written in FORTRAN which is approximately equivalent to the ANSI standard, making it quite portable. The code compiled and executed on all of

the target systems with no changes to the source code. The ADS code proved acceptable to the class of problems demonstrated. ADS allows nonlinear, nonequality constraints, but does not allow nonlinear equality constraints, the power equality constraint which is fundamental to the problem of optimized propfan blade design was approximated by two nonequality constraints as shown below:

$$C_{p,load} - \left(C_{p,req} + \varepsilon\right) \leq 0 \tag{3-13}$$

$$\left(C_{p,req} - \varepsilon\right) - C_{p,load} \leq 0 \tag{3-14}$$

where $C_{p,req}$ is the required power as supplied by the engine, $C_{p,load}$ is the calculated power loading on the blade, and ε is a small allowable error term. Generally, good results were obtained by the ADS code. It is important to note that the optimization library is a component in the MSO computing environment. It is replaceable, just as are the analysis codes.

For the aerodynamic analysis in our demonstration problem (described in Section 3.3), we used the in-house VORP code as described in Section 3.3.1. For structural response computations required in the multidisciplinary analysis, we used the implicit finite-element analysis code, NIKE3D, from Lawrence Livermore National Laboratories. This code can be used for both static and unsteady analyses, including the modal analyses necessary to evaluate the flutter phenomenon to be explored in Phase II. For the stochastic analysis executed in Phase I, we used STOFES, ARA's Stochastic Finite Element analysis System. For this computation, Monte-Carlo Simulation (MCS) is used for probabilistic analysis and NIKE3D is used for the finite element analysis. Within STOFES, NIKE3D is simply a module which can be replaced. For instance, STOFES can also drive the DYNA3D explicit finite-element code.

The demonstration problem was developed as a serial application, and then converted into a portable parallel application by parallelizing the code at various levels. Primary software development was performed on ARA's HP730 workstation, with only minor porting issues (unrelated to the parallel code) being resolved on the Intel iPSC/860 and the LACE cluster. The parallel implementation of the demonstration problem represents a subset of the MSO methodology and consists of the following components: 1) Parallel minimization/optimization using the NASA ADS optimization code, 2) Single-level parallel aerodynamic performance analysis using the ARA VORP code; and 3) Multi-level parallelization of the optimization and aerodynamic analyses. Serial implementations of the demonstration problem for multidisciplinary optimization and probabilistic analysis were also performed.

3.4.2 Parallelization of the Demonstration Problem

Overview. 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. Job-level parallelism is the most coarse-grained

form of parallelism. It can be achieved by starting multiple instances of a complete parallel program on different processors, or by starting multiple parallel programs on different processors. Sub-program parallelism is achieved by grouping subtasks into recursive subroutines, where each concurrent instance of the recursive subroutine contains a unique copy of all local variables. The duplication of local variables is necessary on shared memory machines, to prevent one process from overwriting the calculations of another. It is necessary on distributed memory machines since a process on one processor cannot access any memory on another processor. Loop-level parallelism can be achieved automatically on certain parallel computers, using specialized compilers that perform automatic parallelization at multiple levels. Although these specialized tools exist, they do not facilitate developing portable parallel programs. The Phase I example problem was coded to demonstrate both sub-program and loop-level parallelism in such a way that the code is portable to many computer systems, including multi-processor distributed memory machines, networks of workstations, and multi-processor shared-memory machines.

To guarantee that the parallel code would be portable, all parallelization was implemented using the Parallel Virtual Machine approach; no specialized compilers were used. The following list describes the primary parallelization efforts:

1) Independent calculation of aerodynamic influence coefficients for the VORP code.

2) Independent calculation of sensitivity coefficients for the optimization algorithm.

3) Integrated calculation of aerodynamic influence coefficients and sensitivity coefficients, to study implementation of multi-level parallelization strategies.

When optimizing an aerodynamic shape, such as the propfan blade of the example, an initial geometry is required. The optimization is used to tune the initial shape to optimum performance. Given a good initial guess, it is generally not necessary to update the influence coefficients of a vortex ring method such as VORP on every design loop iteration. The aerodynamic influence coefficients of the VORP code are functions purely of the geometry of the blade. Each element's vortex ring will influence every other element; however, the largest influence on a given element is the influence of its own vortex ring. In fact, as Figure 3-5 suggests, only large changes in the geometry of neighboring elements will noticeably affect the velocity influence at any control point. However, in order to investigate the efficiency of multi-level parallelism which will be necessary for full-scale multi-disciplinary design optimization, the third parallelization study recomputes influence coefficients at every iteration.

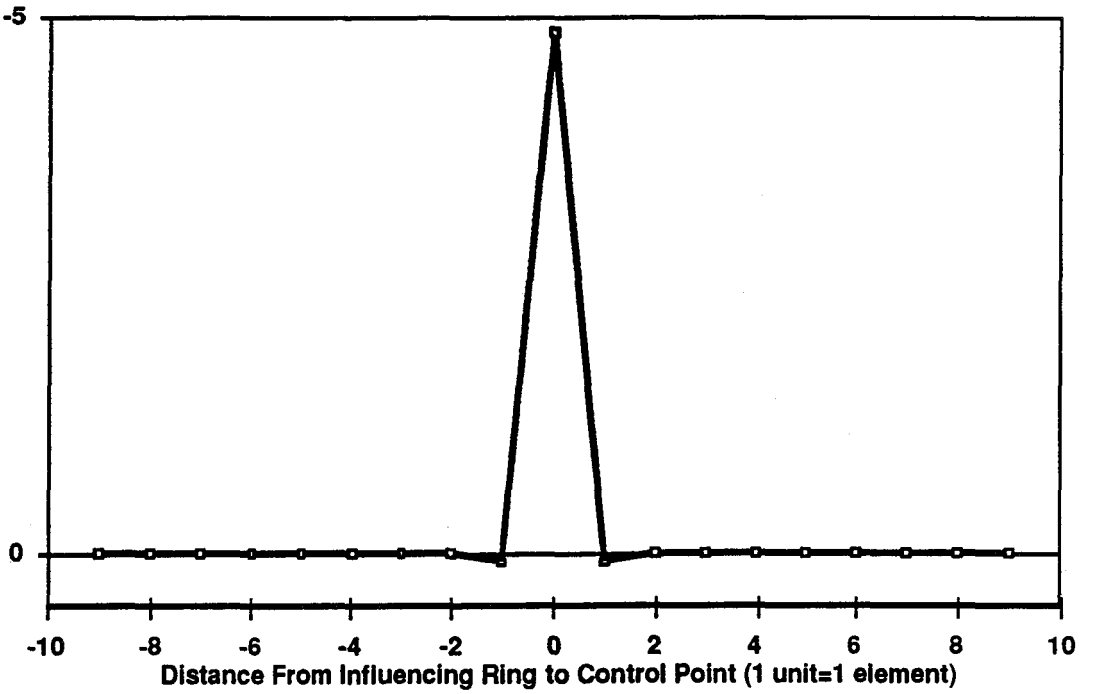

Figure 3-5. Velocity Influence Coefficient of an Element due to Itself and due to Neighboring Elements

Master-Slave Paradigm. The demonstration code was parallelized using a master-slave or client/server paradigm, with the message-passing of PVM as the method of communication between the processes. When the code is executed, the user specifies the number of processes to start. Except on a single-processor workstation, this usually coincides with the number of processors available. Given N processes, one (1) process will act as master, and (N-1) processes will act as slaves. The master process has a variety of jobs which include:

1) Read input data.

2) Initialize all slaves with data from the input file.

3) Execute main optimization loop.

4) Write output data and results.

Since the code must in general execute on distributed memory computers, and since the master is the only process which performs I/O, the master must initialize all slaves by sending messages which contain the input data. The following code segment demonstrates portions of the coding of the slave initialization using PVM calls, in standard FORTRAN. The actual code transmits much more information.

```
1:    integer bufid,msginfo,n_sect,nj,jspace,N,nready
2:    parameter(IMSG_POKEDATA=3)
3:    read(iinput,3) n_sect,nj,jspace
4:    call pvmfinitsend(PVMDEFAULT,bufid)
6:    call pvmfpack(INTEGER2,n_sect,1,1,msginfo)
7:    call pvmfpack(INTEGER2,nj,1,1,msginfo)
8:    call pvmfpack(INTEGER2,jspace,1,1,msginfo)
9:    do i=1,N-1
10:       call pvmfsend(sltids(i-1),IMSG_POKEDATA,msginfo)
11:   end do
12:   nready=0
13:   do while (nready.ne(N-1))
14:     call pvmfrecv(-1,-1,bufid)
15:     if (bufid.gt.0) then
16:       slavetag=0
17:       msgtag=0
18:       call pvmfunpack(INTEGER4,msgtag,1,1,msginfo)
19:       call pvmfunpack(INTEGER4,slavetag,1,1,msginfo)
20:       call ipushslave(slavetag)
21:     end if
22:   end do
```

The first two lines are, of course, variable declarations. The third line reads three variable values out of an input data file. Lines 4 through 8 initialize a PVM send, and pack the data into a "send buffer", which stores the data in the master processor's memory until it is received by all of the slaves. The loop in lines 9-11 sends a message to all the slaves. The loop in lines 13-22 waits for confirmation from all of the slaves. This second loop exists for the following reason; the master must be sure all slaves have been initialized before making requests. In this segment of code (lines 13-22), most of the commands are either familiar, or are PVM functions beginning with the letters **pvmf**.

One function that we created is **ipushslave()**. This function was created to make it easier to track a large number of slave processes. In PVM, every process has a unique identification number. When a process wants to send a message containing data, it must direct the message to the identification number of the destination process. The function **ipushslave()** takes a single argument which is the identification number of a slave process. The slave is then placed in a queue within the master process. From this point, whenever the master needs to make a request, it will call another function, **ipopslave()**, to retrieve a slave identification number, which will be removed from the queue. If no slaves are available, **ipopslave()** will return an error until a slave becomes available. While waiting for slaves, the master will sit in a loop waiting for results. Every time the master receives a message, it will again call **ipushslave()**, so that the slave queue is continuously filled and emptied. These two functions allow the master process to easily keep the slaves busy, even when processors of differing computational power are used. In order to make efficient use of the hypercube architecture, the code was restructured

so that the master process as well as the slaves would run on the cube. The method of starting the code is to use a separate program to spawn the processes on the cube. The startup code looks like:

```
1: parameter(MSG_YOUAREMASTER=1,MSG_SIMPACK=2)
2: integer N,tids(100),nalloc
3:     call pvmfspawn('propfan',PVMDEFAULT,'*',N,tids,nalloc)
4: if (nalloc.le.0) then
5:      call pvmfexit()
6:      stop
7: end if
8: call pvmfinitsend(PVMDEFAULT,bufid)
9: call pvmfpack(INTEGER4,MSG_YOUAREMASTER,1,1,msginfo)
10: call pvmfpack(INTEGER4,int(nalloc-1),1,1,msginfo)
11: do i=1,(nalloc-1)
12:   call pvmfpack(INTEGER4,int(tids(i)),1,1,msginfo)
13: end do
14: call pvmfsend(tids(0),MSG_YOUAREMASTER,msginfo)
15: N=1
16: call pvmfrecv(-1,MSG_SIMPACK,bufid)
17: do while (N.ne.nalloc)
18:   call pvmfrecv(-1,MSG_SIMPACK,bufid)
19:   N=N+1
20: end do
21: write(*,*) 'The propfan code has started successfully'
```

Here, line 3 attempts to allocate the process **propfan** on a cube of N nodes. If the spawn succeeds, lines 8-13 initializes a message buffer to send to the master, and line 14 sends the message. The loop in lines 11-13 provide the master process with a list of slave processes. The master is responsible for notifying these process that they are slaves. Once a slave has received the master's message, it sends notification to the host. Once the host receives as many notification messages as it originally allocated, it prints a message to the screen, and terminates, leaving the optimization to run entirely on the iPSC/860.

Parallel Computation of Aerodynamic Influence Coefficients. To illustrate the implementation strategy consider the following segment of code, which is the loop used to calculate the aerodynamic influence coefficients in parallel. Pseudo-code has been substituted in places to make the segment easier to read.

```
1:   integer i,j,k,n_blades,nelem,bufid,slavetag,msgtag
2:   integer msginfo
3:   i=1
4:   n_msg_sent=0
5:   do while (i.le.nelem)
6:     j=1
7:     do while (j.le.nelem)
8:       [set current blade row to 0]
9:       k=1
10:      do while (k.le.n_blades)
11:        okay_incr_k=0
12:        call pvmfnrecv(-1,-1,bufid)
13:        call pvmfunpack(INTEGER4,slavetag,1,1,msginfo)
14:        call ipushslave(slavetag)
15:        [process results from slave]
16:        n_msg_sent=n_msg_sent-1
17:        sendtid=ipopslave()
18:        if (sendtid.gt.0) then
19:          [initialize message buffer for send]
20:          call pvmfsend(sendtid,MSG_CALCHRIF,msginfo)
21:          if ([element j is a trailing edge element]) then
22:            okay_incr_k=1
23:          end if
24:          n_msg_sent=n_msg_sent+1
25:        end if
26:        if (okay_incr_k.eq.1) then
27:          [set blade row=next blade row]
28:          k=k+1
29:        end if
30:      end do
31:      j=j+1
32:    end do
33:    i=i+1
34:  end do
```

This code contains an inside loop which accounts for multiple blades. Since the solution is calculated only for a single blade, only axial flow is modeled and all blades must have the same loading. The loop above simultaneously distributes requests to the slaves and processes results as they arrive. However, after all the requests have been distributed, the loop ends. In order to synchronize with the slaves, a second loop is required. This loop gathers results which arrive after the first loop has completed.

```
1:   do while (n_msg_sent.gt.0)
2:     call pvmfrecv(-1,-1,bufid)
3:     if (bufid.gt.0) then
4:       call pvmfunpack(INTEGER4,msgtag,1,1,msginfo)
5:       call pvmfunpack(INTEGER4,slavetag,1,1,msginfo)
6:       call ipushslave(slavetag)
7:       [process results from slave]
8:       n_msg_sent=n_msg_sent-1
9:     end if
10:  end do
```

The calculation of the velocity influence of a quadrilateral vortex ring around one element on the control point of another element is relatively fine-grained, with approximately 420 floating point operations performed per influence coefficient. The influences due to the helical wake vortex filaments require a numerical integration, and these calculations are more coarse-grained. While there are many more quadrilateral influences than helical influences, the computation can still be dominated by the helical influences (depending on the discretization used to compute the helical influences). For the example problem here, the combination results in a medium-grained problem.

Parallel Computation of Optimization Sensitivity Coefficients. A general sensitivity coefficient is defined by the equation:

$$S_i = \frac{\partial F}{\partial x_i} \tag{3-15}$$

where F is a response function, and x_i is the i^{th} design variable. For the current design vector, x_i, i=1,N, S_i can be approximated by the finite-difference equation:

$$S_i \approx \frac{F(x_1, x_2, x_i + \delta, ..., x_N) - F(x_1, x_2, x_i + \delta, ..., x_N)}{2\delta} \tag{3-16}$$

where δ is a small percentage of the initial design variable value. This formulation of S_i requires two function evaluations, and these are calculated in parallel. Thus, for N design variables, one calculation of all sensitivity coefficients requires 2N function evaluations in parallel. The coding of the sensitivity coefficients use essentially the same approach as for the aerodynamic influence coefficients; however, the parallel speedup is greater since the sensitivities are much more coarse-grained.

Multi-Level Parallelism. For the multi-level parallelism the master process distributes requests to perform function evaluations to clusters of N nodes. Within a cluster, one process performs the function evaluation, and uses the remaining N-1 nodes to evaluate influence coefficients in parallel. In distributing a single function evaluation request to a cluster, the master process will select one node to be the sub-

master. The master sends a list of all nodes in the cluster along with other requisite information, as indicated in the code below:

```
1:   sendtid=ipopslave()
2:   if (sendtid.gt.0) then
3:    call pvmfinitsend(PVMDEFAULT,bufid)
4:    call pvmfpack(INTEGER4,MSG_CALCOBJP,1,1,msginfo)
5:    call pvmfpack(INTEGER4,num_nodes,1,1,msginfo)
6:    do i=1,num_nodes
7:     call pvmfpack(INTEGER4,ipopslave(),1,1,msginfo)
8:    end do
9:    [pack additional information for problem]
10:    call pvmfsend(sendtid,MSG_CALCOBJP,msginfo)
11:  end if
```

The interesting lines in this code are lines 5-8. These distribute a cluster of nodes. These nodes together calculate a single function evaluation using the parallel influence coefficient evaluation code from above. Thus, function evaluations and influence coefficients are calculated in parallel. In performing multi-level parallelism, the ipopslave () and ipushslave utility routines are not efficient, because they only allow slaves to be grouped arbitrarily. When a unique problem is performed on a cluster of nodes, the cluster should contain neighboring nodes as much as possible, depending on the hardware topology. For example, it is inefficient to perform a single task on a cluster of nodes from multiple cubes in a hypercube computer. In Phase II additional utility routines will be created which generate clusters of nodes which are neighbors on the hardware topology.

3.5 PARALLEL PERFORMANCE

3.5.1 Description of the Timing Studies

On both the Intel iPSC/860 and the LACE Cluster, timing studies were based on wall clock time. On conventional, multi-user computers such as the LACE Cluster machines, accurate timing studies are difficult to achieve, due to varying degrees of network traffic and competition for CPU time. However on the Intel, network traffic is limited to messages passed between the cube and the workstation which allocated a cube. There is no multi-user competition for either CPU time or network communications. Therefore wall clock time is consistent from run to run and yields a good indication of the true parallel speedup. Intel supplies sophisticated Parallel Performance Analysis Tools (PAT) profiling tools with their systems; however, these tools only operate on Sun workstations, and could not be used on the NASA-Ames Silicon Graphics host workstations. Even when such tools are available, they do not necessarily represent timing data accurately. The Intel tools are *post-mortem* tools, meaning the timing data is acquired and stored to disk during the execution of a program. The data must be analyzed after the program has completed execution in

order to be accurate. However, disk I/O is sufficiently slow that it can dramatically change the program execution and adversely affect post-mortem timing studies.

On the LACE Cluster, timing studies were performed during peak usage hours, as well as during dedicated time. In both cases, all timing runs were repeated several times in order to acquire an average time. This is important for two primary reasons. During peak usage hours, there are many users competing for CPU time and network time. This means the state of any given machine is unknown. It is possible and likely that certain machines will be heavily loaded while other machines are lightly loaded, but it is impossible to know which. The only way to achieve reasonable time measurements is to take an average. During dedicated time, there is no competition for CPU time. The measured timings were repeated consistently during dedicated time, with variations no larger than 0.5%.

We performed several implementation and timing studies including investigation of the multi-level parallel decomposition approach that will be necessary for achieving massive parallelism and to achieve high efficiency for problems with the large memory requirements typical of multi-disciplinary problems. The list below summarizes the implementation studies executed in Phase I:

1. ***Parallel Optimization, Intel iPSC/860.*** Parallel computation of sensitivity coefficients used in aerodynamic shape optimization of the advanced propfan blade. This first study is used as a benchmark and to confirm expectations of high parallel efficiency for coarse-grained analyses, using from one to twenty processors.

2. ***Parallel Aerodynamic Analysis, Intel iPSC/860.*** Parallel computation of aerodynamic influence coefficients to obtain loads on the propfan blade. This study investigates the feasibility of achieving high parallel efficiency for a finer-grained problem. Analyses are executed using from one to fifty processors.

3. ***Parallel Optimization, IBM RS/6000 workstation network.*** Repeat of study described under item 1 for the workstation network. Here we investigate the portability of the PVM toolkit and study parallel efficiency over a workstation network, both in a dedicated mode and in normal operation mode, using from one to twenty workstations.

4. ***Multi-level Parallelism, Intel iPSC/860.*** Simultaneous parallel computation of both sensitivity coefficients and influence coefficients. This study investigates the feasibility of simultaneously exploiting more than one level of parallelism (which will be necessary for achieving large scale speedup for practical problems of interest). We use a top-down approach and exploit the coarsest grained part of the problem first (the sensitivity coefficients) and use remaining available processors for the finer grained part of the problem (the influence coefficients). Analyses are executed using from ten to forty processors.

5. ***Multi-disciplinary Optimization, HP 9000/730 Workstation.*** Coupled aeromechanical optimization of the advanced propfan blade. An improved optimization procedure is made possible since the blade shape can be optimized starting from the cold shape as opposed to a presumed hot shape. The purpose of this study is to investigate the feasibility of performing multi-disciplinary optimization, determine computational resources required, and to identify special requirements that will be needed for parallelization in Phase II.

6. ***Stochastic Analysis HP, 9000/730 Workstation.*** Stochastic structural analysis of the propfan blade under load was executed. Parallelization was not performed since the feasibility of parallel probabilistic analysis has been demonstrated in earlier research. The purpose was to demonstrate the feasibility of stochastic analysis for the example problem.

3.5.2 Phase I Results

The results of the investigations described above are presented here. As described earlier the example problem selected for the feasibility investigations is the optimum design of an advanced propfan blade. Figure 3-6 shows a typical propfan blade and the loadings it encounters. For the Phase I studies we considered only airloads.

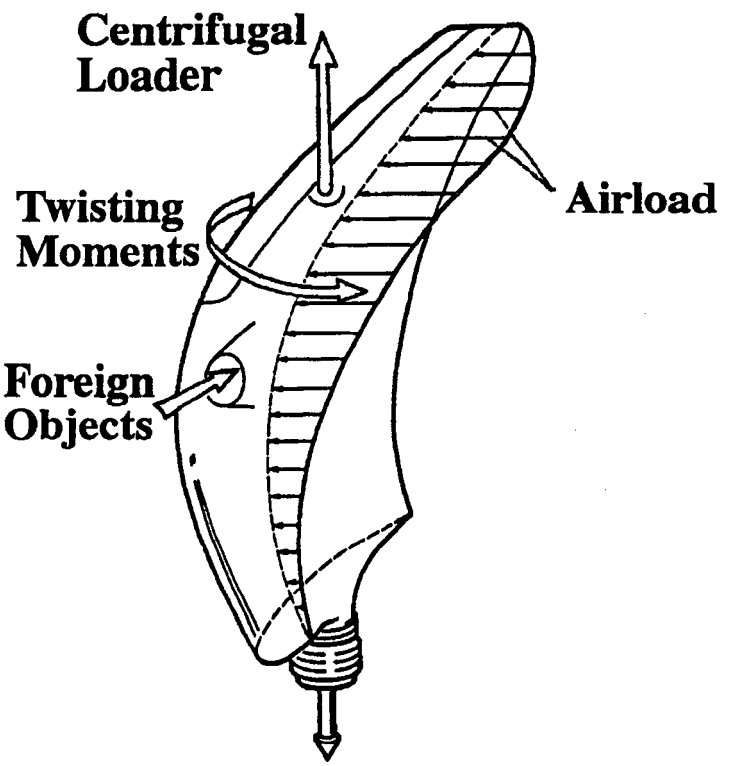

Figure 3-6. Typical Propfan Blade and Loads

Parallel Optimization, Intel iPSC/860. The general aerodynamic results are presented in Figures 3-7, 3-8, and 3-9. Figure 3-7 shows the twist distributions for the SR7-blade (With $\beta_{0.75radius}$=38°) and for the two optimized blades. It is seen that there is little difference between the SR-7 and the optimized shape at the same power coefficient (Cp=0.94). This indicates the SR-7 twist distribution is nearly optimal within this single-constraint problem. The optimized shape at a higher power coefficient is essentially a shift of the SR-7 twist, with only small changes in the distribution. There is larger change near the tip of the blade, and this is an expected result, as the outer blade (around the 75% radius station) contributes the largest thrust increment. Figure 3-8 shows the thrust versus advance ratio for the three blade shapes plus experimental data for the SR-7 blade (Mach=0.165, $\beta_{0.75radius}$=38°) [Aljabri, 1987]. The VORP code thrust in the vicinity of the design advance ratio J=0.9 is close to the experimental data. At advance ratios greater than about 1.3, the experimental data and VORP results deviate due to viscous effects such as flow separation which are not modelled by VORP. There is virtually no difference between the SR-7 and the optimized shape at the same power coefficient (Cp=0.94). However, the optimized shape at the higher power coefficient provides more thrust as expected. Figure 3-9 shows propeller efficiency versus advance ratio. As expected, for the same power coefficient, the SR-7 and optimized shapes have nearly identical efficiency distributions, though the optimized shape has a slightly larger range of effective advance ratios and a higher maximum efficiency. The optimized shape at a higher power has slightly increased efficiency and a wider range of effective advance ratios. It is important to make clear the fact that this Phase I problem is a subset of the full propfan design problem. A full optimization to include the sweep distribution (blade section stacking) as design variables and flutter analyses would potentially produce an optimized shape that is significantly different than the SR-7.

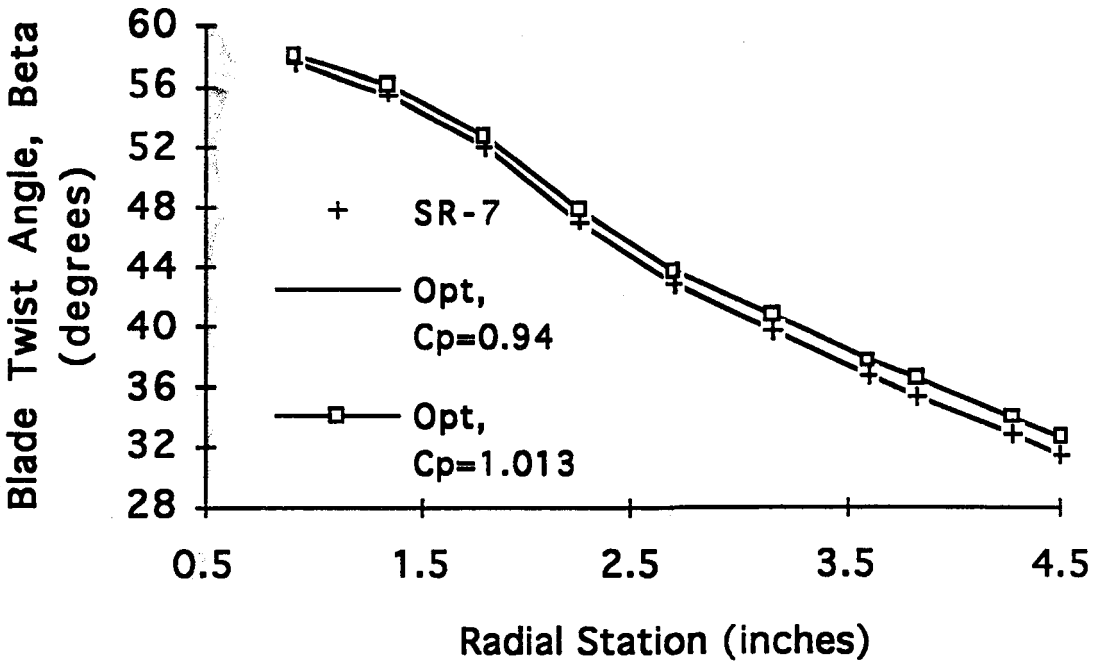

Figure 3-7. Blade Twist Distributions

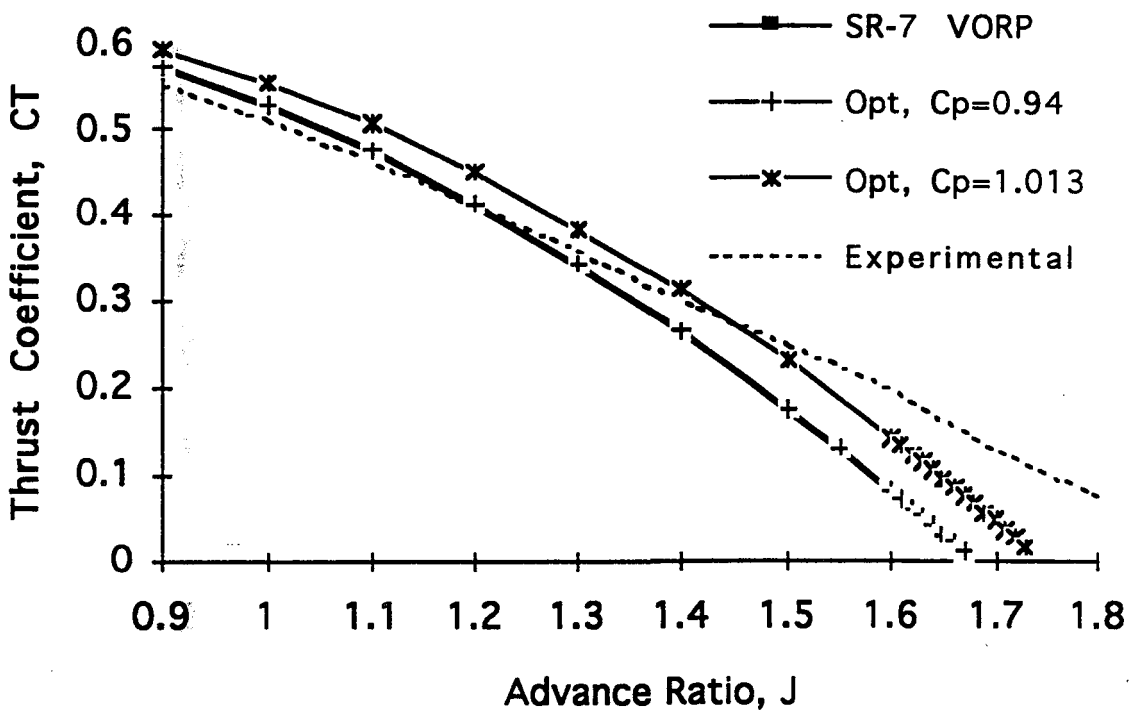

Figure 3-8. Thrust Versus Advance Ratio

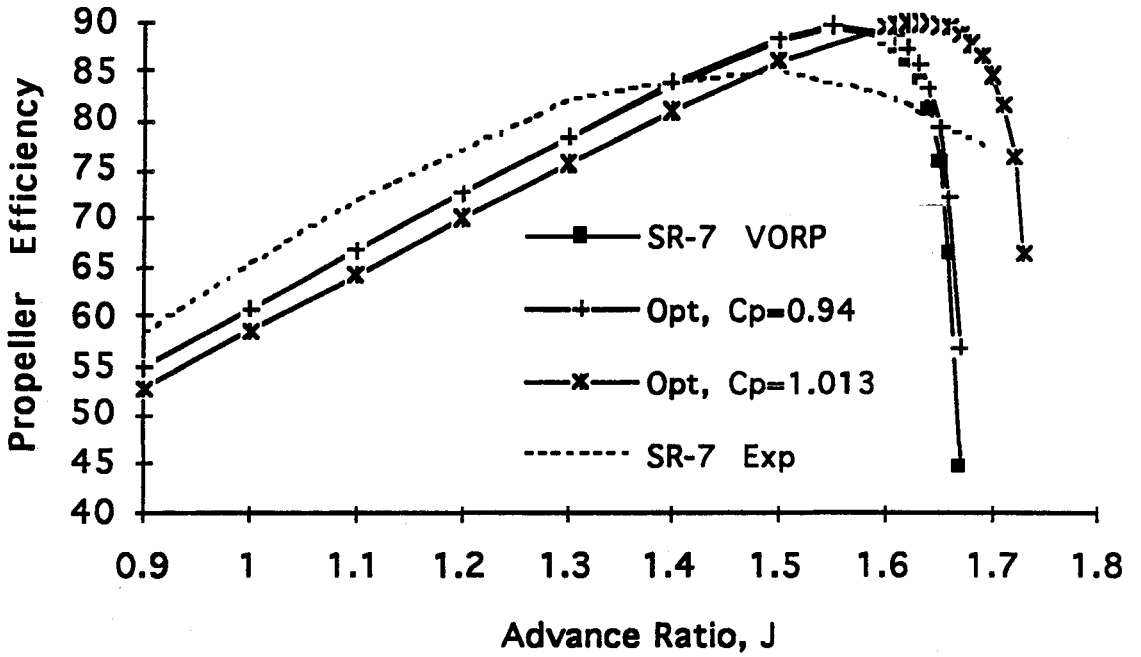

Figure 3-9. Propeller Efficiency Versus Advance Ratio

A study of coarse-grained parallelism on the Intel iPSC/860 was performed by calculating optimization sensitivity coefficients in parallel. Figure 3-10 shows parallel speedup versus number of slave processors. Nearly perfect linear speedup of almost 19 times is achieved for 20 slave processors. However we note that 19 processors are no more effective than 10 because we have 20 optimization sensitivity coefficients to compute in parallel. Hence for 19 processors, 19 sensitivities are computed in parallel and the 20th is computed using one processor while the other 18 processor are idle. To obtain higher speedup for 11 through 19 processors, multi-level decomposition is required.

Considering processor idling that is unavoidable for single-level parallelism, we can compute the maximum theoretical efficiency that can actually be obtained versus the ratio NSOL/NPROC, where NSOL is the number of independent parallel computations (20 in this case) and NPROC is the number of processors as

$$\varepsilon_T = \frac{NSOL/NPROC}{INT\left(\frac{NSOL}{NPROC}\right) + I\left[MOD\left(\frac{NSOL}{NPROC}\right) > 0\right]} \tag{3-17}$$

In Figure 3-11, Eq. 3-17 is plotted and the measured data are superimposed on the plot. It is clear that we have in fact achieved close to maximum possible speedup (for single level parallelism). The difference from theoretical speedup here is due to communication overhead.

The results for this coarse-grained study are positive. Every multidisciplinary optimization problem includes such coarse-grained parallelism and can expect similar speedup results.

Parallel Aerodynamic Analysis, Intel iPSC/860. In order to study finer-grained parallelism on the Intel, a matrix of aerodynamic influence coefficients necessary for the aerodynamic analysis was calculated in parallel using ARA's VORP code. As described in Section 3.4.2, we characterize this problem as medium-grained. Figure 3-12 shows typical blade surface pressure coefficients, normalized to the free stream velocity plus the radial component at the panel control point. These actually are differential pressures between the upper and lower surfaces, since VORP only models the flow over the camber surface of the blade.

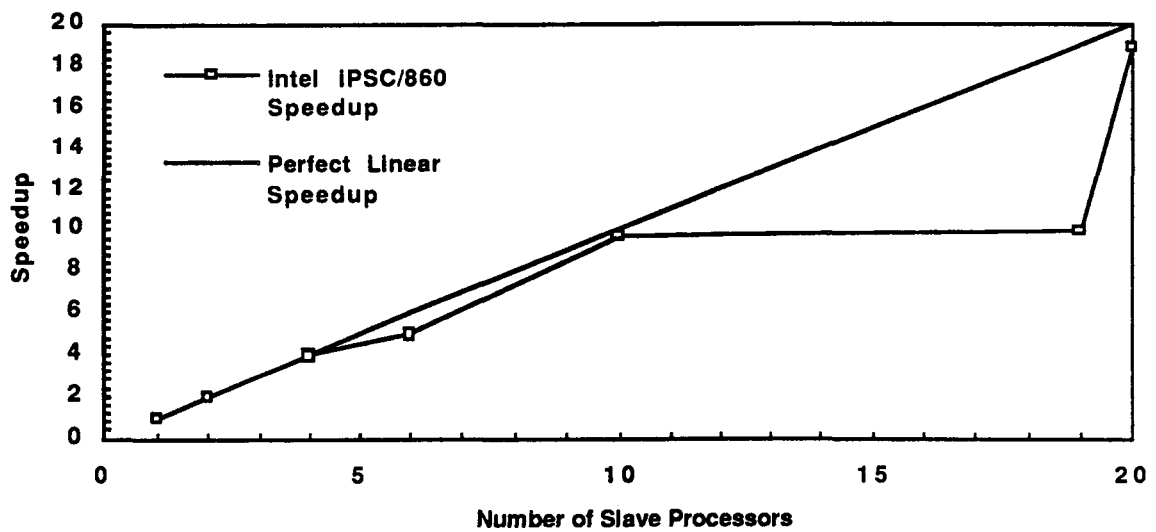

Figure 3-10. Parallel Speedup for Sensitivity Coefficient Calculations on the Intel

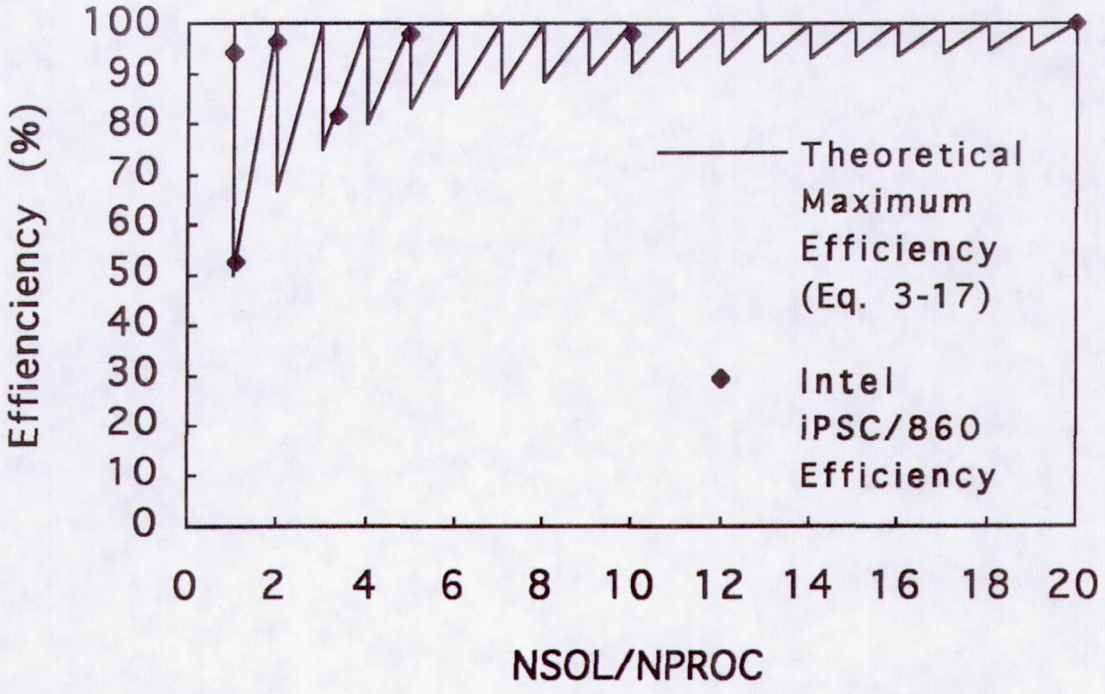

Figure 3-11. Theoretical Maximum Parallel Efficiency for Single-Level Parallelism

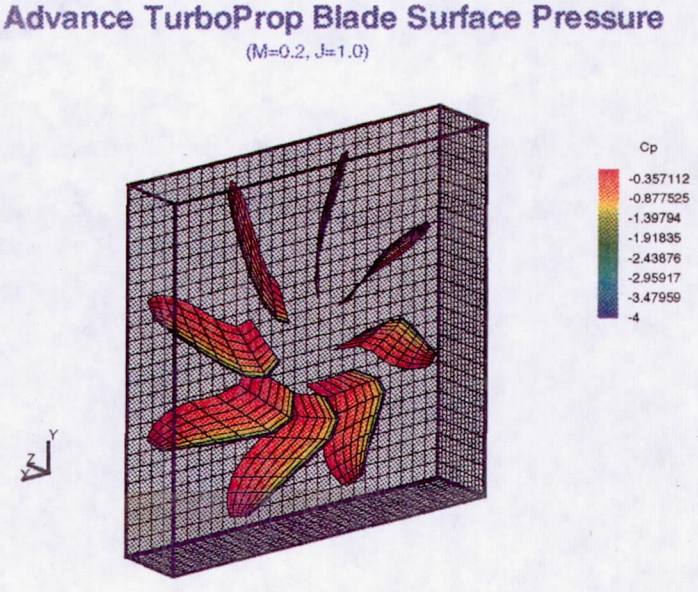

Figure 3-12. Advanced Turboprop Blade Differential Surface Pressures

Figure 3-13 shows the parallel speedup on the Intel, and Figure 3-14 shows the efficiency. High parallel efficiencies are again achieved, exceeding 75% for 31 slave processors and roughly 60% for 50 processors. Although the efficiency is not as high as for the coarse-grained parallelism of the previous example (as expected) we still achieve a speedup of 24 times for 31 processors and 30 times for 50 slave processors. Also, there are no flat portions in this speedup curve, as there were with the calculation of the sensitivity coefficients, since a 10x10 grid of elements requires NSOL=80000 influence coefficients for an 8-bladed propeller. This yields high values of NSOL/NPROC for any number of slave processors, resulting in negligible processor idling and maximum theoretical efficiencies close to 100%. The drop-off in efficiency in going from 31 to 50 slave processors is because this problem contains both medium-grained (helical influence coefficients) and fine-grained (quadrilateral influence coefficients) computations. As the number of processors is increased the finer-grained computations have a greater impact on overall efficiency. In future research we will investigate approaches for more efficiently parallelizing the fine-grained part of the problem (e.g., computing groups of influences at a single node and using element-by-element solution procedures that don't require global matrix assembly).

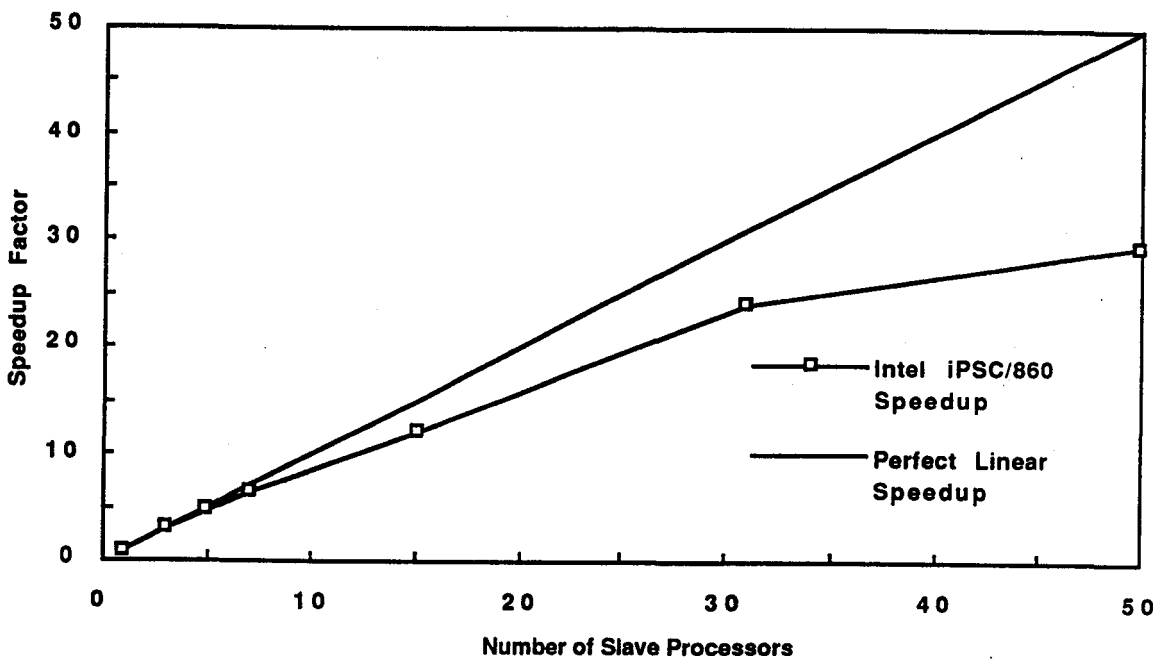

Figure 3-13. Parallel Speedup for Influence Coefficient Calculations on the Intel

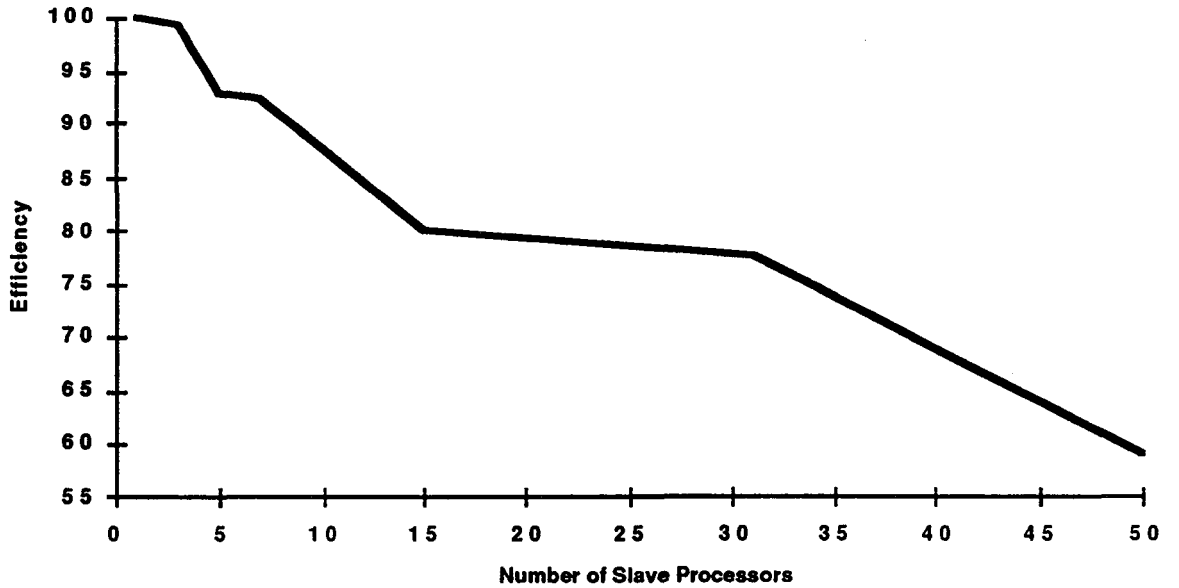

Figure 3-14. Parallel Efficiency for Influence Coefficient Calculations on the Intel

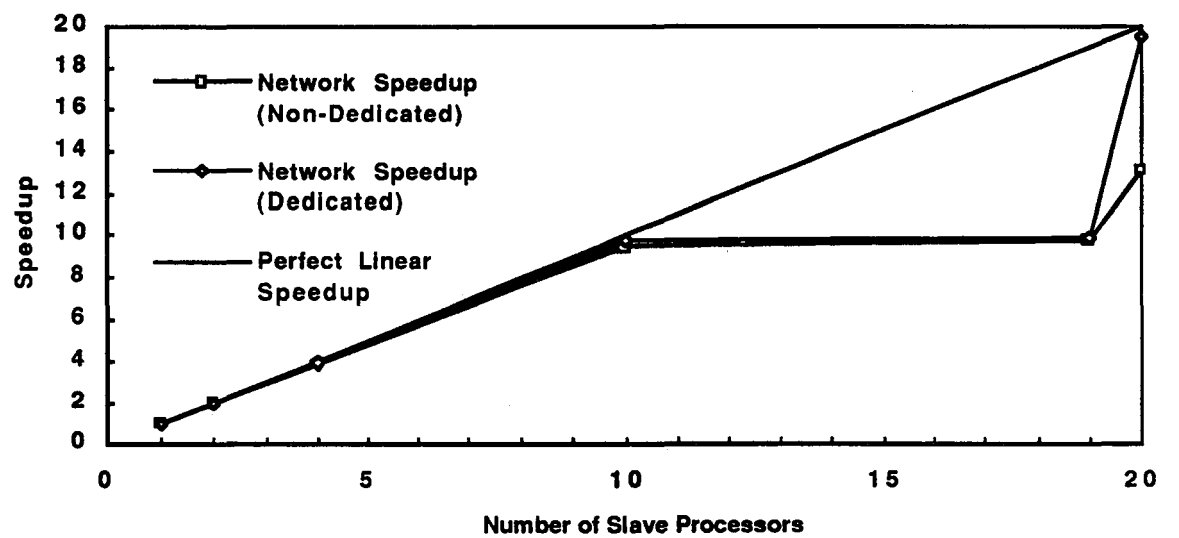

Figure 3-15. Parallel Speedup for Sensitivity Coefficient Calculations on the IBM RS/6000 Network

Parallel Optimization, IBM RS/6000 workstation network. The parallel optimization timing studies performed on the Intel were repeated on the NASA Lewis LACE Cluster, which contains 32 IBM RS/6000 workstations. The speedup for the coarse-grained sensitivity coefficients is shown in Figure 3-15. A large Ethernet network

such as the one which links the 32 IBM's together will in general have very high network traffic and communications overhead. Additionally, during normal work hours, there will be varying degrees of competition for CPU time, and different workstations will have different loads. These factors induce losses in efficiency, and the periods of heavy usage and for dedicated time. Even during the dedicated time, network traffic is not guaranteed to be negligible, since the physical connections may require Ethernet packets between the nodes be redirected through other systems which could be heavily loaded. However, the LACE Cluster is configured such that 24 of the IBM workstations are connected together on a subnet which is separated from other systems, in order to minimize network traffic during dedicated runs. For our dedicated timing studies we used the subnet. Notice that the dedicated results exhibit essentially perfect linear speedup at 20 slave processors (of course, the speedup is flat from 10 through 19 processors as explained for the Intel timing studies). For dedicated time, efficiencies are as good on the network for this coarse-grained application as on the Intel. This is a very important result, since networks of workstations are so widely available. Also important is the fact that the coarse-grained speedup is quite good even when the network is heavily loaded. Finally, a key result is that the source code as tested was the same on the LACE Cluster and on the Intel hypercube, demonstrating the portability of the PVM approach.

Multi-level Parallelism, Intel iPSC/860. Multi-level parallelism will be needed to effectively use large numbers of processors on massively parallel machines and to alleviate memory/processor demand. For example, for optimization problems, if only a single level of parallelism is used, it is only possible to effectively use a number of processors equal to the number of sensitivity coefficients (20 in our example). Also, for large problems it is necessary to distribute computation of an individual sensitivity coefficient over several processors to reduce memory demand per node. Without such a distribution, secondary storage would need to be used which can eliminate parallel speedup [Sues et al. 1993].

In order to investigate the parallel efficiency of multi-level parallelism, an optimization was solved with the influence coefficients re-calculated at every iteration in parallel, simultaneously with the sensitivity coefficients. Hence, processors are allocated in clusters with each cluster of processors performing one sensitivity analysis and each node in the cluster evaluating a group of influence coefficients.

Four separate cases were timed. For each case we used 10 clusters and varied the number of slave processors per cluster from one to four.

Figure 3-16 shows the multi-level parallelism timing study results. First, we observe that multi-level parallelism exacts an overhead cost. For ten slave processors the speedup with multi-level parallelism is roughly 80% of the speedup when only a single level of parallelism is exploited. However, single level parallelism can only keep 20 slave processors busy (since we have 20 sensitivity coefficients to compute for this example), thus, speedup reaches a plateau. By using multi-level parallelism we can effectively use more processors. At forty slave processors the multi-level speedup is

approximately 3.2 times the multi-level speedup at ten slave processors. Thus, we still have an 80% efficiency. Of course, we can also keep large numbers of processors busy simply by invoking only influence coefficient parallelism. However, the multi-level parallelism has a coarser granularity than single-level influence coefficient parallelism. Thus, efficiencies are greater for the multi-level parallelism.

Multi-disciplinary Optimization, HP 9000/730 Workstation. A multi-disciplinary design optimization was performed on a single workstation by coupling our aerodynamic loads analysis code, VORP, with the NIKE3D implicit finite element code. The blade finite element model is a 10x10 grid of 4-node Hughes-Liu Shell elements. In the subroutine which evaluates the objective thrust, the following functions are performed in order: 1) Generate cold shape geometry for current iteration, 2) Execute the NIKE3D implicit finite-element code given the aerodynamic loads on the cold shape, 3) Re-evaluate the aerodynamics for the deformed, "hot" shape, to determine the current value of thrust and begin the next design interaction. Theoretically, it is necessary to iterate these three steps to obtain the true hot shape before beginning the next design iteration. However, in practice a single pass at each design iteration is sufficient since an exact hot shape is only necessary at the final design step to validate the optimum solution. By performing this three-step analysis of the objective function, the final optimized shape is the cold, undeformed shape. Hence we directly obtain the manufactured shape of the blade. This multi-disciplinary approach provides a significant time savings over a purely aerodynamic shape optimization (that

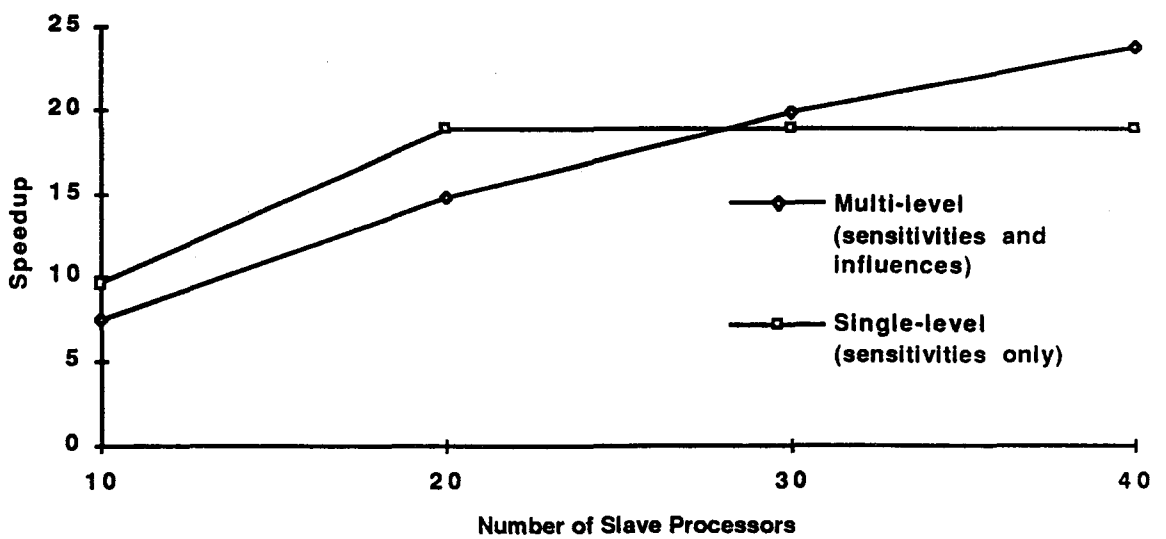

Figure 3-16. Speedup Results for Multi-Level Parallelism on the Intel: PSC/860 (Ten Clusters Used for Each Multi-level Study)

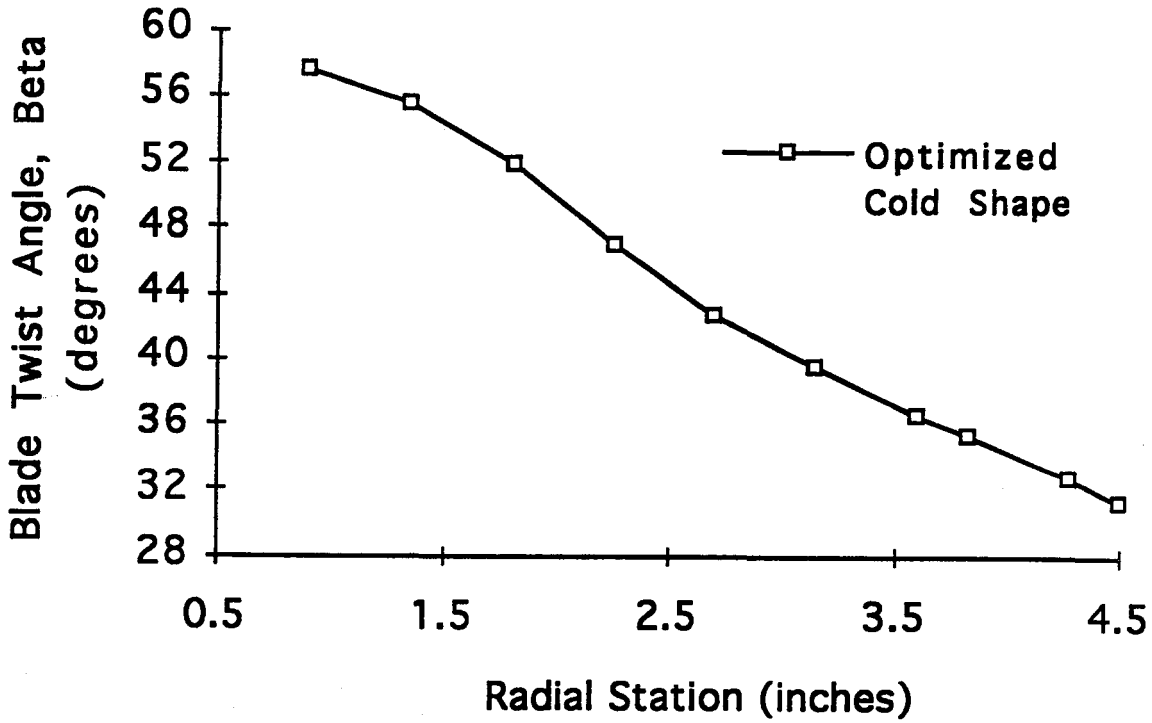

Figure 3-17. Optimized Cold Shape Twist Distribution

is, uni-discipline optimization). In the uni-discipline optimization the structural engineer must back-calculate the cold shape after the optimization is completed. The multi-disciplinary approach provides the solution in one step. Figure 3-17 shows the linear twist angle distribution used as the initial guess and the final undeformed cold shape of the optimized blade. Figure 3-18 shows the final deformed "hot" shape from the final NIKE3D analysis.

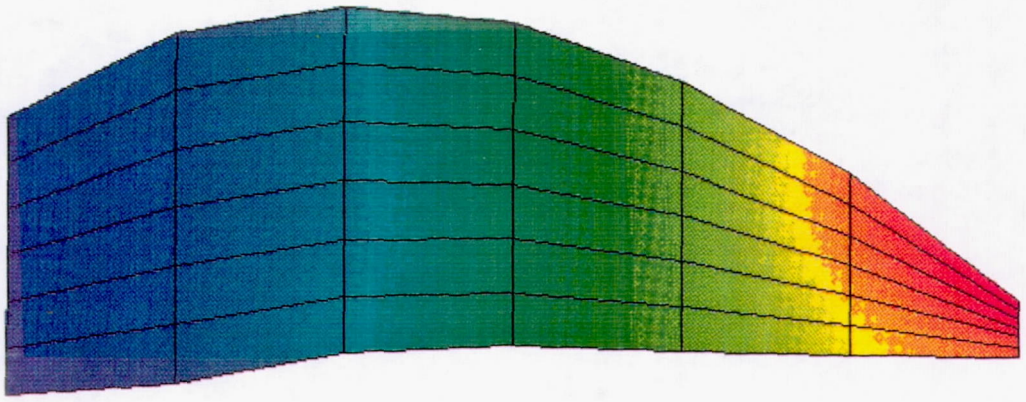

Figure 3-18. Deformed "Hot" Shape

Stochastic Analysis, HP 9000/730 Workstation. A stochastic analysis of the propfan blade was executed considering uncertainties in the blade material properties and thickness. Table 3-1 shows the random variable modeling. The stochastic analysis was executed using STOFES, ARA's stochastic finite element analysis system. For this computation, Monte-Carlo Simulation (MCS) is used for probabilistic analysis and NIKE3D is used for the finite element analysis. Figure 3-19 shows the Cumulative Distribution Function (CDF) curve for the tip deflection of the blade in the z direction. The response statistics are listed in Table 3-2. The 1000 sample MCS requires approximately 30 minutes to execute on the workstation. Given the near linear speedup that has been demonstrated for MCS in earlier studies [Sues et al. 1993], MCS can be the preferred method for stochastic optimum design on a parallel computer.

TABLE 3-1. STATISTICAL PARAMETERS FOR STOCHASTIC ANALYSIS OF THE PROPFAN BLADE

Variable	Mean Value	Coefficient of Variation (%)	Distribution
Young's Modulus (*psi*)	1.0E+07	10.0	Lognormal
Poisson's Ratio	0.33	1.7	Uniform
Thickness (*in*)	0.78	5.0	Lognormal

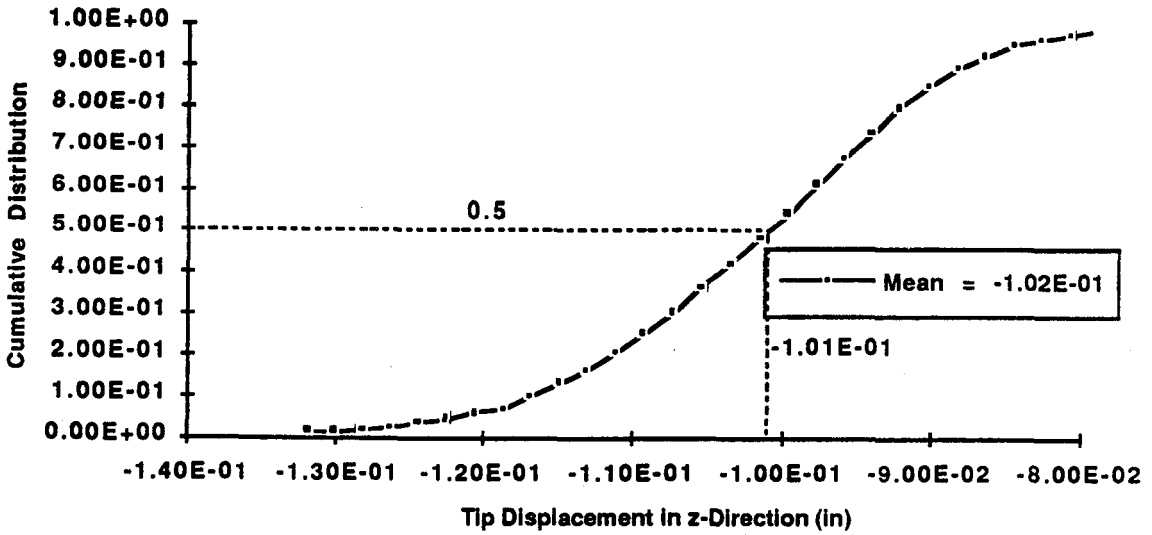

Figure 3-19. Cumulative Distribution Function for Tip Displacement in the Z-Direction

TABLE 3-2. RESPONSE STATISTICS

Variable	Mean Value	Standard Deviation	Coefficient of Variation (%)
Tip-displacement in the Z-direction	-0.102	0.0114	11.2

CHAPTER 4

SUMMARY, CONCLUSIONS, AND RECOMMENDATIONS

4.1 SUMMARY

The objective of this Phase I research was to formulate a parallel multi-disciplinary stochastic optimization (MSO) methodology and establish the feasibility of achieving efficient parallel implementation on distributed-memory parallel computers and on a network of workstations. Further, we set an objective to demonstrate that the parallel implementation would be portable so that recoding of the parallel instructions would not be necessary for the two platforms. As demonstrated in the previous chapters, both efficiency and portability goals were achieved. These are the key technical objectives that must be met in order to demonstrate the potential of parallel design optimization tools to meet the very high performance challenges that have been posed for 21st century aircraft.

Conventional approaches to design are inefficient when dealing with complex problems that involve many different engineering disciplines, and for problems that involve new technologies for which little practical experience exists such as the design of advanced propulsion systems or HSCT vehicles. With advanced analysis and numerical optimization techniques, engineers and scientists are attempting to reduce the need for costly trial and error approaches and reduce the amount of testing and experimentation required. The computational cost of these multi-disciplinary optimizations can be quite large. However, the advent of parallel and distributed computing offers the potential to significantly reduce the computer time required for multi-disciplinary design optimizations. Also, by using parallel and distributed computing, engineers can now consider aspects of a design which were previously ignored. For example, the treatment of uncertainties in design optimization has long been recognized as important. Both design and constraint variables with a large degree of uncertainty can significantly affect the optimum design. In addition, in pushing performance limits it is crucial that aircraft reliability be quantified.

Parallel processing applications, in general have been hampered by issues of code portability and reformulation, and hardware availability. Hence, our goal for this research is to overcome these hurdles. To meet this goal engineers will need tools that can efficiently and automatically decompose the problem in a way that considers the hardware on which the problem is to be solved and the specific problem characteristics. To demonstrate the feasibility of developing such a tool and meet the Phase I objectives we performed several investigations, summarized below.

First, we investigated portable parallel programming paradigms to identify the approach best suited to meet our overall objectives. We selected the message passing paradigm as implemented in Parallel Virtual Machine (PVM) Version 3.1. This proved to be an excellent choice for both portability and parallel efficiency.

Next, we formulated the multi-disciplinary stochastic optimization methodologies in order to identify sources of parallelism and identify specific areas of investigation for the Phase I feasibility study. This research combined with some of our earlier studies also culminated in refining a comprehensive multi-level computational strategy to exploit these sources of parallelism in a way that minimizes memory/processor requirements while also minimizing parallel overhead.

Finally, we performed several implementation and timing studies including investigation of the multi-level parallel decomposition approach that will be necessary for achieving massive parallelism and to achieve high efficiency for problems with the large memory requirements typical of multi-disciplinary problems. The example problem that was selected for the feasibility investigations is the optimum design of an advanced propfan blade. The following parallel implementation and timing studies were executed in Phase I: (1) Parallel computation of sensitivity coefficients used in aerodynamic shape optimization of an advanced propfan blade on the Intel iPSC/860 using from one to twenty processors; (2) Parallel computation of aerodynamic influence coefficients to obtain loads on the propfan blade on the Intel iPSC/860 using from one to fifty processors; (3) Parallel computation of sensitivity coefficients used in aerodynamic shape optimization on a network of IBM RS/6000 workstations using from one to twenty workstations; (4) Multi-level parallel computation of both sensitivity coefficients and influence coefficients on the Intel iPSC/860 using from ten to forty processors; (5) Coupled aeromechanical multi-disciplinary optimization of the advanced propfan blade on an HP 9000/730 workstation; and (6) Stochastic structural analysis of the propfan blade on an HP 9000/730 workstation.

4.2 CONCLUSIONS AND RECOMMENDATIONS

The investigations in Phase I demonstrate that it is possible to effectively parallelize the key computational elements of stochastic multi-disciplinary optimization problems. Nearly perfect linear speedup was achieved for the coarse-grained sensitivity coefficient computations on both the Intel and on the workstation network (speedup of almost 19 times for twenty slave processors). Very high parallel efficiencies were also achieved for the finer-grained aerodynamic influence coefficient computations on the distributed-memory Intel iPSC/860 (75% for thirty-one processors and 60% for fifty processors). These high core efficiencies allowed for high parallel efficiency in the multi-level decomposition implementation. The feasibility investigations also demonstrate the portability and high parallel performance of the Parallel Virtual Machine library. All code in this Phase I research was developed and tested on a single HP 9000/730 workstation. The code was then ported to both the Intel and the workstation network with no modifications to the PVM portions of the code. PVM was also demonstrated to provide the level of functional control required to implement the multi-level parallelism needed to achieve large scale parallelism on massively parallel hardware.

To achieve large scale parallelism and reduce memory/processor demand, multi-level parallel decomposition strategies along with specially designed computational algorithms are needed. For typical design optimization problems, if only a single level

of parallelism is used, it will not be possible to keep very large numbers of processors busy. Also, for the large MSO problems that are of practical interest, it is necessary to distribute computations over several processors to reduce memory demand per node (or use computational algorithms that minimize memory requirements). As a simple example, if 96 Mbytes of storage are required to solve a structure and only 16 Mbytes are available at each processor node, 6 processors at a minimum must be assigned to solve a single structure. Decomposition among these 6 processors must then be accomplished. Thus, a two-level decomposition for stochastic optimization would use clusters of 6 processors each to perform independent sensitivity analyses or Monte-Carlo simulation histories.

Based on the Phase I studies, we can draw the following conclusions and recommendations:

1. Near linear speedup can be achieved on workstation networks for the coarse-grained parallelism encountered at the top level of MSO problems.

2. Massively-parallel supercomputers can achieve high parallel speedup even on medium to fine-grained problems and are well suited to exploiting multi-level parallelism.

3. The Parallel Virtual Machine (PVM) library is a proven solution for portable parallel programming and provides the functional control and parallel efficiency needed to effectively implement multi-level parallel MSO.

4. There are several inherent levels of parallelism in multi-disciplinary, stochastic optimization (MSO) problems, and these must be taken advantage of to fully exploit the potential of parallel computing in aeropropulsion system design.

5. A generalized MSO code should be portable across a wide range of architectures, including networks of low-cost workstations, in order to increase its commercial appeal.

6. Parallel control algorithms must be developed to automatically decompose a problem and exploit the multiple levels of parallelism for MSO problems, to make parallel execution commercially viable.

7. Specially adapted computational algorithms should be developed for efficient parallel implementation in order to reduce memory requirements and processor idling.

8. Hybrid-memory architectures, consisting of an interconnection of shared-memory processor nodes (four to eight processors that share memory at a node) will likely be optimal for parallel MSO problems. This architecture maps directly to the multiple levels of both coarse and fine grained parallelism exhibited by MSO problems. This is an emerging technology and is typified by the massively parallel Intel Paragon machine (which

now has more than 30 installations worldwide and is currently being installed at NASA/Ames), networks of Silicon graphics multi-processor workstations, and the NASA Hypercluster machine.

The rapid advances that are occurring in parallel hardware (including hybrid-memory architectures), availability of large amounts of high-speed memory at very small cost, and commonplace occurrence of workstation networks will clearly make parallel computing a widely accessible tool. It remains to develop the innovative computational algorithms that can automatically and efficiently exploit the parallelism that exists in engineering design problems.

REFERENCES

Aljabri, A. S., 1987. "Wind Tunnel Tests on a One-Foot Diameter SR-7L Propfan Model," AIAA-87-1892, *AIAA/SAE/ASME/ASEE 23rd Joint Propulsion Conference*, June 29-July 2, San Diego, California.

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

Bain, W. L., 1990. "Aggregate Distributed Objects for Distributed Memory Parallel Systems," *Proceedings of the Fifth Conference on Distributed Memory Concurrent Computers*, Charleston, SC, April 9-12.

Bertin, J. J. and M. L. Smith, 1979. *Aerodynamics for Engineers*, Prentice Hall, New Jersey.

Blech, R. A. and E. J. Milner, 1992. "Turbomachinery CFD on Parallel Computers," *Symposium on High-Performance Computing for Flight Vehicles*, Washington, DC.

Carriero, N. and Gelernter, D., 1989. "How to Write Parallel Programs: A Guide to the Perplexed," work supported by National Science Foundation SBIR Grant ISI-8704025, Department of Computer Science, Yale University, New Haven Connecticut.

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

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.

Goldstein, M. A., 1929. "On the Vortex Theory of Screw Propellers," *Proceedings of the Royal Society of London*, Series A, Vol. CXXIII, London.

Hager, R. D. and D. Vrabel, "Advanced Turboprop Project," NASA SP-495, pp. 13-14.

Karamcheti, K., 1966. *Principles of Ideal-Fluid Aerodynamics*, Robert E. Krieger Publishing Company, Malabar, Florida.

Liu, P-L. and A. Der Kiureghian, 1986. "Multivariate Distribution Models with Prescribed Marginals and Covariances," *Probabilistic Engineering Mechanics*, Vol. 1, No. 2, pp. 105-112.

Liu, P-L., and A. DerKiureghian, 1991. "Finite Element Reliability of Geometrically Nonlinear Uncertain Structures," *Journal of Engineering Mechanics*, Vol. 117, No. 8, August.

Liu, W. K., et al., 1987. "Finite Element Methods in Probabilistic Mechanics," *Prob. Eng. Mech.*, Vol. 2, No. 4.

Madsen, H. O., et al., 1986. *Methods of Structural Safety*, Prentice-Hall, Englewood Cliffs, NJ.

Mahadevan, S., 1992. "Reliability-Based Optimization Using Sequential Quadratic Programming," Pres. at *Prob. Mech. and Struc. and Geotech. Reliability ASCE Conf.*, Denver, CO, July.

Nataf, A., 1962. "Determination des Distribution dont les Marges sont Donnees," *Comptes Rendus de l'Academie des Sciences*, Paris, 225, pp. 42-43.

PVM 3 User's Guide and Reference Manual, 1993. ORNL/TM-12187, Oak Ridge National Laboratory, Oak Ridge, Tennessee.

Rackwitz, R. and B. Fiessler, 1978. "Structural Reliability Under Combined Random Load Sequences," *Comput. Struct.* 9, 489-94.

Rhodes, G. S., P. L. Coe, Jr., and J. N. Perkins, 1991. "30 × 60 Foot Wind Tunnel Test Highlights For An Over-The Tail Advanced Turboprop Configuration," Presented at the *29th Aerospace Sciences Meeting*, AIAA 91-0681, January 7–10, Reno, Nevada.

Schueller, G.I., et al., 1989. "On Efficient Computational Schemes to Calculate Structural Failure Probabilities," *Prob. Eng. Mech.*, Vol. 4, No., 1, March.

Spector, Alfred Z., "Multiprocessing Architectures for Local Computer Networks", August, 1981.

Sues, R. H., Y. J. Lua, and M. D. Smith, 1993. "Parallel Computing for Probabilistic Fatigue Analysis," *Proceedings of the 34th AIAA SDM*, La Jolla, California.

Sues, R. H., H-C. Chen, and F. M. Lavelle, 1992. "The Stochastic Pre-Conditioned Conjugate Gradient Method," *Probabilistic Engineering Mechanics*, Volume 7, pp. 175-182.

Sues, R. H., H-C. Chen, C. C. Chamis, and P. L. N. Murthy, 1992. "Programming Probabilistic Structural Analysis for Parallel Processing Computers," *AIAA Journal*, Volume 30, Number 12.

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

Sues, R.H., Y-K Wen, and A.H-S Ang, 1985. "Stochastic Evaluation of Seismic Structural Performance, *Jour. of Struc. Eng.*, ASCE, Vol. III, No. 6, June.

Theodorsen, T., 1948. *Theory of Propellers*, McGraw-Hill Book Company, Inc., New York.

Twisdale, L. A., R. H. Sues, and C. E. Murphy, 1988. *Assessment of Reliability-Based Design Methodology for Prot. Structures*, ESL-TR-38-27, AFESC, Tyndall AFB, FL, Dec.

Wu, Y-T, 1987. "Demonstration of a New, Fast Probability Integration Method for Reliability Analysis," Advances in Aerospace Structural Analysis, AD-09 (*Proc. Symp. on Prababilistic Structural Design & Analysis*, Winter Annual Mtg., ASME, Miami Beach, FL, 17-22 Nov. 1985); and *Jour. of Eng. for Ind.*, ASME, New York, Feb.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (*Leave blank*)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	October 1994	Final Contractor Report

4. TITLE AND SUBTITLE

Portable Parallel Stochastic Optimization for the Design of Aeropropulsion Components

5. FUNDING NUMBERS

WU-324-01-00
C-NAS3-26839

6. AUTHOR(S)

Robert H. Sues and G.S. Rhodes

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Applied Research Associates, Inc.
6404 Falls of Neuse Road, Suite 200
Raleigh, North Carolina 27615

8. PERFORMING ORGANIZATION REPORT NUMBER

E-8725

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-195312

11. SUPPLEMENTARY NOTES

Project Manager, Dale A. Hopkins, Structures Division, organization code 5210, NASA Lewis Research Center, (216) 433-3260.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Unclassified - Unlimited
Subject Category 39

12b. DISTRIBUTION CODE

13. ABSTRACT (*Maximum 200 words*)

This report presents the results of Phase I research to develop a methodology for performing large-scale Multi-disciplinary Stochastic Optimization (MSO) for the design of aerospace systems ranging from aeropropulsion components to complete aircraft configurations. The current research recognizes that such design optimization problems are computationally expensive, and require the use of either massively parallel or multiple-processor computers. The methodology also recognizes that many operational and performance parameters are uncertain, and that uncertainty must be considered explicitly to achieve optimum performance and cost. The objective of this Phase I research was to initialize the development of an MSO methodology that is portable to a wide variety of hardware platforms, while achieving efficient, large-scale parallelism when multiple processors are available. The first effort in the project was a literature review of available computer hardware, as well as a review of portable, parallel programming environments. The second effort was to implement the MSO methodology for a problem using the portable parallel programming language, Parallel Virtual Machine (PVM). The third and final effort was to demonstrate the example on a variety of computers, including a distributed-memory multiprocessor, a distributed-memory network of workstations, and a single-processor workstation. Results indicate the MSO methodology can be well-applied towards large-scale aerospace design problems. Nearly perfect linear speedup was demonstrated for computation of optimization sensitivity coefficients on both a 128-node distributed-memory multiprocessor (the Intel iPSC/860) and a network of workstations (speedups of almost 19 times achieved for 20 workstations). Very high parallel efficiencies (75% for 31 processors and 60% for 50 processors) were also achieved for computation of aerodynamic influence coefficients on the Intel. Finally, the multi-level parallelization strategy that will be needed for large-scale MSO problems was demonstrated to be highly efficient. The same parallel code instructions were used on both platforms, demonstrating portability. There are many applications for which MSO can be applied, including NASA's High-Speed-Civil-Transport, and advanced propulsion systems. The use of MSO will reduce design and development time and testing costs dramatically.

14. SUBJECT TERMS	15. NUMBER OF PAGES
Optimization; Probability theory; Structural analysis; Structural reliability; Aerodynamics; Parallel programming	68
	16. PRICE CODE: A04

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	