2005

An application of parallel computation to Collaborative Optimization

Shahab Nayyer
Louisiana State University and Agricultural and Mechanical College

AN APPLICATION OF PARALLEL COMPUTATION TO COLLABORATIVE OPTIMIZATION

A Thesis
Submitted to the Graduate Faculty of the
Louisiana State University and
Agricultural and Mechanical College
in partial fulfillment of the
requirements for the degree of
Master of Science in Industrial Engineering

In
The Department of Industrial Engineering

by

Shahab Nayyer
B. Tech, Indian Institute of Technology-Chennai, 2002
May, 2005

Acknowledgements

I would like to express my sincere gratitude to my major adviser Dr. Charles McAllister for his insight and guidance at all stages of my research. He has always been helpful and ready to answer all my questions.

I would also like to thank Dr. Thomas Ray, Dr. Hassan Mashriqui, and Dr. Xiaoyue Jiang for serving on my committee and critically evaluating my work. I am especially thankful to Dr. Ray and Dr. Mashriqui for providing financial assistance during the period of my work.

My parents and family members have been the source of courage and strength. It has all been possible because of their constant encouragement and prayers. I am also grateful to my friend Vikas for helping me in innumerable ways during this research. I would like to thank everyone who has directly or indirectly contributed to my work.

Last but not the least I would like to thank almighty for everything in life.

Table of Contents

List of Tables

List of Figures

Abstract

Multidisciplinary Design Optimization (MDO) has evolved as a discipline which provides a body of methods and techniques to assist engineers in solving large scale design problems. There are many frameworks for formulating MDO problems. These frameworks can be broadly classified as single-level or bi-level formulations. Collaborative Optimization (CO) is one of the popular bi-level formulations to solve an MDO problem.

There are numerous design optimization problems which are highly CPU time intensive and require a long simulation time. With the advent of cheaper and faster available PC's, distributed parallel computer clusters have become very popular. These clusters provide large computing power and can be used to solve problems faster and more efficiently. This research is an attempt to take advantage of the computational power of parallel computers in the field of design Optimization. The robust design optimization of an Internal Combustion Engine has been formulated using CO and implemented using parallel computers. Considerable savings in Wall Time has been achieved. A generic strategy for solving similar problems has also been devised. A benchmarking program has also been developed to assess theoretical speedup for any problem size. This program uses the Collaborative Optimization framework and simulates a design optimization on distributed memory clusters.

1. Introduction

In any engineering system there are interactions among the physical phenomena and the hardware parts. These interactions make the synergistic whole larger than sum of its parts. Multidisciplinary Design Optimization (MDO) [31] has evolved as a field which provides a body of methods and techniques to assist engineers in solving large scale design problems. There are numerous design optimization problems which are highly CPU time intensive and require a long simulation time. With the advent of cheaper and faster personal computers, distributed parallel computer clusters have become very popular. There are some real life problems which take many years of computation time. The enormous computing power provides the ability to solve these problems in a fast and efficient manner. This research is an attempt to take advantage of the computational power of parallel computers in the field of design Optimization [14].

Collaborative Optimization (CO) is used to solve large scale optimization problem in engineering. CO is implemented by breaking larger problems into smaller problems and these smaller problems are linked with each other by various design, function or performance constraints. Once formulated, these problems can be solved by the one of the optimization routines available e.g., GAMS or DSIDES. A CO problem is solved using DSIDES by performing the system and subsystem level optimizations sequentially. In problem formulations where variables between the different subsystems are not dependent on each other, the system and subsystem level optimization can be performed simultaneously. This provides a great opportunity for the application of parallel computation to Collaborative Optimization. This approach is likely to reduce the

total simulation time. It will also allow scientists and engineers to solve larger and more complex problems in a more realistic time frame.

1.1 Problem Description

McAllister and Simpson have introduced the CO-DSP framework [18]. They formulated a multidisciplinary robust design optimization formulation to evaluate uncertainty encountered in the design process. The formulation is a combination of the bi-level Collaborative Optimization framework and the multi-objective approach of the compromise Decision Support Problem. Their proposed framework was demonstrated with the design of a combustion chamber of an internal combustion engine.

The proposed framework was found to effectively attain solutions that are robust to variations in design variables and environmental conditions. The combustion chamber problem was divided into two subsystem analyses routines, thermodynamics and geometry. Each subsystem has a set of constraint and variables, and there are no coupled variables. The absences of coupled variables provide a great opportunity for parallel implementation of this problem.

As an example, the proposed CO-DSP framework is used to design a combustion chamber of an internal combustion engine [22, 23]. A flat head design as depicted in Figure 1.1 was assumed. The design variables are the cylinder bore (b), compression ratio (c_r), exhaust valve diameter (d_E), intake valve diameter (d_I), and the revolutions per minute at peak power (w). The objective is to minimize the negative specific power, which is equivalent to the original objective of maximizing the brake power per unit

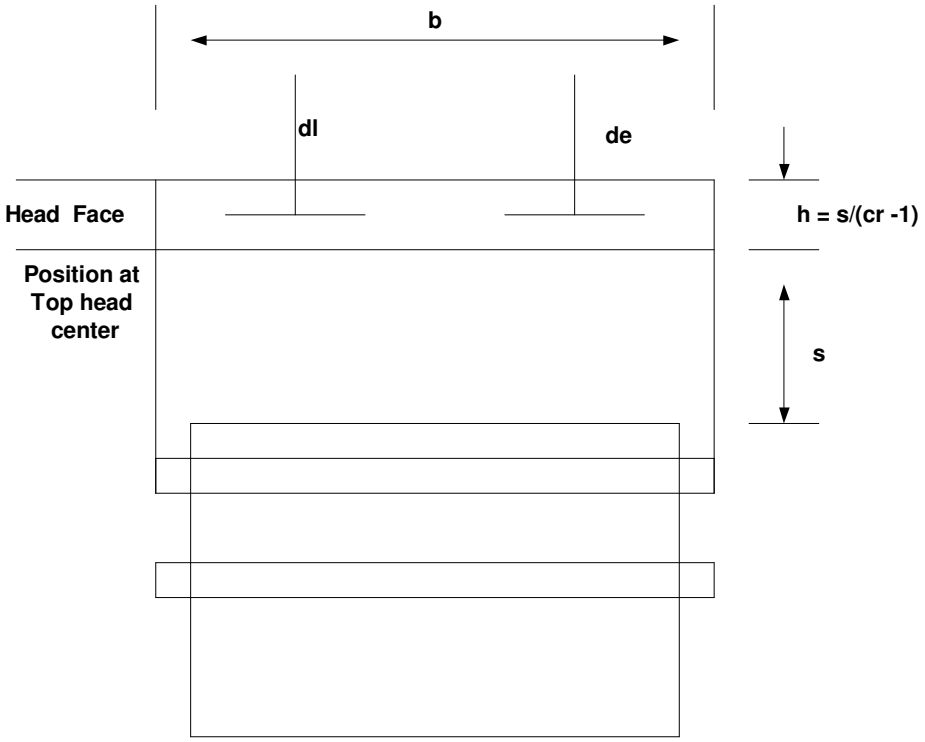

Figure 1.1: Combustion Chamber

engine displacement. The single-level, inequality constrained design formulation presented in [23] follows.

$$\text{Min: } f(x,z) = k_0 w[FMEP - P_0 \eta_t \eta_v] \tag{1.1}$$

where FMEP $= f(c_r, w)$

$P_0 = f(\rho, A_f)$

$\eta_t = f(w, c_r, b)$

$\eta_v = f(w, b, d_I, C_s)$

Subject to:

$$g_1(b) = b - P_1 \le 0 \quad \text{(Min. bore wall thickness)} \tag{1.2}$$

$$g_2(b) = P_2 - b \le 0 \quad \text{(Max. engine height)} \tag{1.3}$$

$$g_3(b, d_E, d_I) = d_I + d_E - K_3 b \le 0 \quad \text{(Valve structure)} \tag{1.4}$$

$$g_4(d_E, d_I) = K_4 d_I - d_E \le 0 \quad \text{(Min. valve diameter ratio)} \tag{1.5}$$

$$g_5(d_E, d_I) = d_E - k_5 d_I \le 0 \quad \text{(Max. valve diameter ratio)} \tag{1.6}$$

$$g_6(w, d_I) = P_3 w - d_I^{\,2} \le 0 \quad \text{(Max. Mach Index)} \tag{1.7}$$

$$g_7(c_r, b) = c_r - 13.2 + 0.045b \le 0 \quad \text{(Knock-limited compression)} \tag{1.8}$$

$$g_8(w) = w - K_7 \le 0 \quad \text{(Max. torque converter rpm)} \tag{1.9}$$

$$g_9(c_r, b) = P_4 - 0.859(1 - C_r^{\,-0.33}) + S_v(C_r, b) \le 0 \quad \text{(Fuel economy)} \tag{1.10}$$

$$X = \text{All variables positive}; 0.7 < b/s < 1.3 \tag{1.11}$$

$$P_0 = \frac{\rho Q}{A_f} \tag{1.12}$$

$$P_1 = \frac{L_1}{K_1 N_c} \tag{1.13}$$

$$P_2 = (4K_2 V / \pi N_c L_2)^{1/2} \tag{1.14}$$

$$P_3 = (9.428)(10^{-5})(4V / \pi N_c)(K_6 C_s) \tag{1.15}$$

$$P_4 = (3.6)(10_6) / K_8 Q \tag{1.16}$$

Where

b = Cylinder bore

c_r = Compression ratio

d_E = Exhaust valve diameter

d_I = Intake valve diameter

w = Revolutions per minute (RPM) at peak power

η_t = Thermal Efficiency

η_v = Volumetric Efficiency

S_v = Surface to volume ratio

s = Stroke of piston

N_c = Number of cylinders

V = Displacement Volume

Q = Lower heating value of fuel

A_f = Air/Fuel ratio

C_s = Port discharge coefficient

ρ = Density of inlet charge

K_i = Parameters and unit conversion

L_i = Block length and height bounds

P_i = Parametric functions

x = Vector of design variables

z = Vector of uncertain parameters

μ_γ = Mean of response

σ_γ^2 = Variance of response

$FMEP$ = Friction Mean Effectiveness Pressure

1.2 Objective

The aims of this research are as follows

1. Formulate the Multidisciplinary Robust Design Optimization solution approach using DSIDES and parallel computers.
2. Implement the design in 1 for a test case for the combustion chamber problem as described by McAllister et al [18].
3. Conduct the benchmarking studies for speed and complexity comparison.

Chapter 2 discusses the some of the underlying concepts in areas related to this research. MDO, Parallel computation, Compromise decision support problems, parallel computation and DSIDES are some of the areas which have been discussed. In Chapter 3, the current state of knowledge in the related field is examined. Chapter 4 discusses in detail the methodology adopted to achieve the objectives of this research. Results of the experiments are presented and discussed in Chapter 5. Conclusions and opportunities of future work are also provided in this chapter.

2. Theoretical Background

This chapter introduces the theory related to this research. Multidisciplinary Design Optimization (MDO) frameworks, Parallel computation and Decision support problems are the main areas of interest and their key aspects are discussed. A comparison is also drawn between the various MDO frameworks and their suitability for parallel implementation.

2.1 Multidisciplinary Design Optimization (MDO)

In the present world almost all engineered and manufactured systems, such as automotive vehicles and aircraft and many consumer products, experience interactions among various physical phenomena and between various components of the full system. These interactions make the system a synergistic whole that is greater than the sum of its parts. A good design should leverage the benefits of this synergy but it is difficult to untangle the web of interactions [31].

The difficulty of these interactions combined with the need to partition the design work into subtasks that can be executed simultaneously in order to compress the project time gave rise to the conventional practice of dividing the detailed design work into specialty areas. This decomposition is centered on a physical phenomenon, such as structural deformations or fluid flow, or on a hardware subsystem, such as a vehicle's suspension system. The technology of MDO evolved to provide a set of techniques that assist engineers in moving a product or process design toward its optimum.

The aerospace industry has been applying optimization in some form to multidisciplinary design problems from the very beginning. The aim of MDO, however, is to provide a more consistent, formalized method for complex system design than is

found in traditional approaches such as parametric trades and sequential iterative design processes. In manufacturing industries, a new product design involves intensive collaboration among teams with specialized disciplines. In the design process, specialized teams often have conflicting considerations, such as thermodynamics, structures and controls, as well as costs and returns on investment. In a car design, a light body might be desirable from a speed point of view, but structurally, it might result in a weak body [13].

A successful design requires manufacturers to integrate the parameters and devise an overall optimum design across disciplines. Often designs are passed between the product teams several times until the differences are minimized and a mutually acceptable solution is found. One of the main challenges in applying MDO to automotive manufacturing is the sophisticated high-fidelity models that have evolved as the standard in the industry. Without superior computing power, elapsed computing time for such detailed models could take years. The computation time needs to be substantially reduced to match to the product design cycle before it can be of any use. With the advent of parallel computation and high-performance computing it is now possible to solve many such problems [14].

The key concept in several of these MDO methods is a decomposition of the design task into subtasks performed independently in each of the modules, and a system-level or coordination task giving rise to a two-level optimization. In general, decomposition was motivated by the obvious need to distribute work over many people and computers to compress the task calendar time. An equally important benefit from the decomposition is granting autonomy to the groups of engineers responsible for each particular subtask in choosing their methods and tools for the subtask execution. As an

additional advantage, the concurrent execution of the subtasks fits well the technology of parallel processing that is now becoming available [14].

The general system optimization problem is stated in the following form:

Given a set of design variables, X
Find : ΔX
Minimize : $\Phi(X, Y(X))$
Satisfy : $G(X, Y(X))$
Bounds on X.

In the above problem, *Y(X)* represents the behavior (state) variables Φ represents the design objective function and G represents the design constraints. A brief description of some of the MDO methods used to solve the system optimization problem is provided in the following sub-sections [16].

2.1.1 All-in-One Method

The All-in-One method is also known as Multidisciplinary Feasibility (MDF) method [7]. It is one of the most popular ways of approaching the solution of MDO problems. In this method, the vector of design variables $\boldsymbol{X}_D$ is provided to the coupled system of analysis disciplines, and a complete multidisciplinary analysis (MDA) is performed via a fixed-point iteration with that value of $\boldsymbol{X}_D$ to obtain the system (MDA) output variable $U(X_D)$ that is then used in evaluating the objective $F(\boldsymbol{X}_D, U(\boldsymbol{X}_D))$ and the constraints $g(X_D, U(X_D))$.

The optimization problem is:

Minimize: $F(X_D, U(X_D))$

Subject to: $g(X_D, U(X_D)) < 0$

and bounds on design variable, $\boldsymbol{X}_D$.

If a gradient-based method is used to solve the above problem, then a complete MDA is necessary not just at each iteration, but at every point where the derivatives are to be evaluated. Thus, attaining multidisciplinary compatibility can be prohibitively expensive in realistic application.

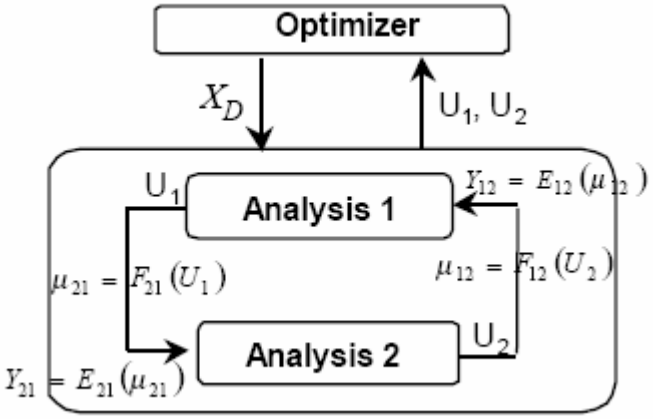

Figure 2.1 A-i-O Model [14]

Figure 2.1 shows the data flow in a A-i-O analysis and optimization. In this Figure, m_{ij} is some spline coefficients obtained using a "fit" F_{ij} of the output of discipline ***j***. F_{ij} may be either an interpolation or an approximation fit. The mapping E_{ij} is an evaluation of the spline representation from discipline ***j*** into a form suitable for use by discipline ***i***.

2.1.2. Individual Discipline Feasible (IDF) Method

The IDF formulation provides a way to avoid a complete MDA at optimization. IDF maintains individual discipline feasibility, while allowing the optimizer to drive the individual disciplines to multidisciplinary feasibility and optimality by controlling the interdisciplinary coupling variables. In IDF, the specific analysis variables that represent

communication between analysis disciplines are treated as optimization variables. They are indistinguishable from design variables from the point of view of a single analysis discipline solver [14].

The IDF formulation is:

$$\text{Minimize}: F(X_D, U(X)) \text{ with respect to } X = (X_D, X\mu)$$
$$\text{Subject to}: g(X_D, U(X)) \leq 0$$
$$C(X) = X\mu - \bar{\mu} = 0$$
and bounds on optimization variable, X.

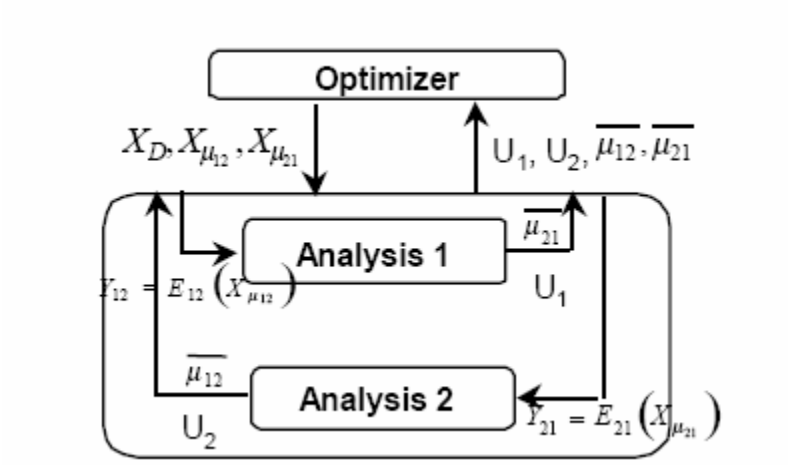

Figure 2.2 IDF Model [14]

X_D is the set of design variables and Xμ is the set of interdisciplinary coupling variables. *C* is referred to as the interdisciplinary constraint. It is important to note that an evaluation of *U(X)* involves executing all the single discipline analysis codes independently with simultaneously available multidisciplinary data *X*. Therefore, the analysis computations can be performed concurrently.

2.1.3. Concurrent Sub Space Optimization (CSSO)

CSSO is a non-hierarchic system optimization algorithm that optimizes decomposed subspaces concurrently. This is followed by a coordination procedure for

directing system problem convergence and resolving subspace conflicts. This corresponds to common design practice where individual design teams optimize their local component designs and compromises are made at the integrated product team or system level.

Each subspace optimization problem is a system level problem formulated with respect to a subset of the total system design vector. Within the subspace optimization, the non local states that are required to evaluate the objective and constraint functions are approximated using the Global Sensitivity Equations (GSE). The CSSO method provides for multidisciplinary analysis feasibility at each cycle but deals with all the design variables simultaneously at the system/coordination problem level [14].

2.1.4. Bi-Level Integrated System Synthesis (BLISS)

The recently introduced BLISS method [31] uses a gradient-guided path to reach the improved system design, alternating between the set of modular design subspaces (disciplinary problems) and the system level design space. BLISS is an A-i-O like method in that a complete system analysis performed to maintain multidisciplinary feasibility at the beginning of each cycle of the path. With BLISS, the general system optimization problem is decomposed into a set of local optimizations dealing with a large number of detailed local design variables (X) and a system level optimization dealing with a relatively small number of global variables (Z) in comparison with the other MDO methods.

2.1.5. Collaborative Optimization (CO)

The CO formulation is a two-level hierarchical scheme for MDO, with the top level being the system optimizer that optimizes on the multidisciplinary variables (or,

system level targets, z) to satisfy the interdisciplinary compatibility constraints (J*) while minimizing the system objective (F). The objective of each subsystem optimizer is to minimize in a least squares sense the discrepancy between the subset of subspace design variables (x_i) and subspace analysis computed responses (y_j) that are common to more than one subspace analysis block and the system level values of these variables, z, while satisfying the subspace constraints (g_j). The system level design variables, z, are considered to be fixed within a subspace problem. A distinction is made between the disciplinary design variables x_{sj}, only of importance to subspace analysis j, and the interdisciplinary design variables x_j, which are common to more than one subspace analysis block.

Like concurrent sub space optimization each subsystem utilizes an independent optimizer complete with disciplinary constraint. The only objective at the subsystem level is to satisfy the compatibility constraints. In contrast to concurrent sub space optimization, collaborative optimization uses a system level optimizer to act on an overall design objective subject to sub system compatibility constraint. The lack of system-level optimization in CSSO provides a significant drawback to applicability. In the design of most engineering systems, there are one or more design objectives. For example an aircraft design problem may be posed to minimize cost and weight and maximize cargo capacity and range. Further more system level optimizer in CO is a method for arbitrating among coupled design variables. If the objective is in terms of one or more of the coupled variables, the corresponding optimal value is selected as the sub system target for succeeding iterations.

The collaborative optimization formulation is intended for cases when the number of disciplinary variables x_{sj} is much larger than the number of interdisciplinary variables x_j. In other words, this formulation is intended for solving design problems with loosely coupled analyses of individually large dimension. Figure 2.3 shows the data flow in a CO analysis and optimization.

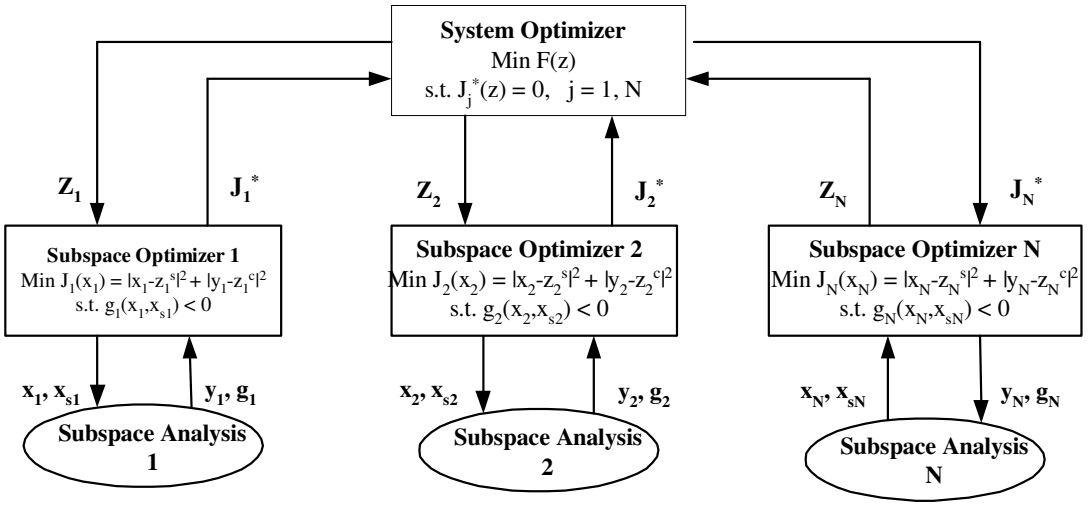

Figure 2.3 Collaborative Optimization Model [14]

2.2 Comparisons of Multidisciplinary Design Optimization Frameworks

McAllister, C D. [19] has compared the various MDO frameworks with respect to various properties. The order of the aspects does not follow any perceived importance.

Aspects 1 and 2 – A good multidisciplinary design optimization framework should provide an environment that facilitates the aggregation of design rules. AAO requires considerable effort to collect and assemble necessary design problem into an optimization package. In contrast, Concurrent Subspace optimization (CSSO) and Collaborative Optimization (CO) implementation is more difficult due to the addition of

compatibility constraint and strategies to manage disciplines. Simultaneous Analysis and Design (SAND) is an intermediary approach that shares the data segregation drawback of AAO but is somewhat more difficult than AAO to implement due to the data stream between the high fidelity codes and the optimization engine.

Table 2.1 Comparison of various MDO frameworks

	Framework			
Desired Properties	**AAO**	**SAND**	**CSSO**	**CO**
1. Easy to assemble design rule	No	Somewhat	Yes	Yes
2. Easy to Implement	Yes	Somewhat	No	No
3. Quick formulation time	Somewhat	Somewhat	Somewhat	Somewhat
4. Permits subsystem contributing analyses	No	Yes	Yes	Yes
5. Fully disaggregated subsystem	No	No	Yes	Yes
6. System-level optimization	Yes	Yes	No	Yes
7. Sub system level convergence	No	No	Yes	No/yes
8. Stable convergence	Yes	Yes	No	Somewhat
9. Quick solution time	Yes	Somewhat	No	No
10. Easy ported for parallel computation	No	Somewhat	Yes	Yes
Total number of desired properties	4	3	5	6

Aspect 3- The total formulation time indicated by the sum of the times required for assembling design rules and their implementation should be of short duration. Generally all framework are compatible, with time spent aggregating design rules offset by reduced implementation time.

Aspect 4 – The framework should allow execution of high fidelity computer codes at resident locations using resident platforms in resident languages. AAO fails by requiring centralization of all parameters, codes, constraints, and objectives. SAND, CSSO, and CO readily incorporates high fidelity nodes.

Aspect 5 – Subsystem should be fully disaggregated to facilitate formulation and experimentation, that is contributing analyses should be incorporated (Aspect 3) in

addition to subsystem-specific design rules. AAO and SAND fail to meet this property because all design rules are considered as a single unit. CSSO and CO provide delineation by subsystem.

Aspect 6 – System level optimization should be provided as a mechanism for achieving design objectives. Unlike the other framework, CSSO does not provide this feature.

Aspect 7 – Subsystem level optimization should be implemented as a means to achieve local subsystem design objectives. CSSO offers this property and multi objective sub system level optimization for CO has been developed by McAllister, C.D. [19]

Aspect 8 – The framework should result in stable convergence. For many classes of optimization problems this is true of AAO and SAND. CSSO has been shown to have poor convergence properties stemming from the fully disaggregated subsystem (Aspect 4) but lack of system level strategy for arbitration of discrepancies among subsystems (Aspect 5). Convergence cannot be proven in general for CO, and as observed in section 3.1.5, some authors report convergence difficulty for poorly behaved mathematically functions. However, for common design problems, CO is widely applied with minimal observance of convergence issues.

Aspect -9 the framework should be easily exploited through optimization to provide quick solution. AAO and SAND are successful in this regard while CSSO and CO requires substantially more computation time to ensure subsystem compatibility.

Aspect 10 - the framework should provide the flexibility for implementation using parallel computation. This aspect is direct converse of Aspect 8. The centralized aspect of AAO forces a single analysis loop. SAND can take advantage of some parallelization

through the separation of high fidelity code. Having fully disaggregated subsystem (Aspect 5), CSSO and CO can make full use of parallel computation.

2.2.1 Selection of CO

McAllister, C D. [19] has cited a number of reasons for his selection of Collaborative Optimization as the framework. The selection of CO for the basis is motivated by several metrics of comparison. CO offers the highest number of desired properties. From the standpoint of this research CO, Aspect 10 is of particular interest. Collaborative Optimization can be readily implemented using parallel computation. A solution approach using Collaborative Optimization requires high solution time. Some of the high solution time can be recovered by using parallel computes. CO being a hierarchical framework can accommodate the formulation of design rules and implementation of optimization approaches at the system and subsystem levels, Aspect 1 and 3-6. The negative aspects of CO are minimized as problem size increases. The high implementation cost of CO is offset by the ease of assembling design rules. For large problems, the time saved in problem formulation is greater than the increased solution time. Hence the greatest benefit of CO is realized by large scale applications. The net formulation time requirement is equivalent for all compared frameworks. Solution times can be further reduced using parallel computers because parallelization of larger problem is more efficient [19].

2.3 Compromise Decision Support Problems

Compromise Decision support problem is a multi-objective decision model which is a hybrid formulation based on mathematical Programming and Goal Programming (Mistree et al. [3]). It is similar to goal programming in the sense that multiple objectives

are formulated as system goals and the deviation function is solely a function of the goal deviation variables. This is in contrast to traditional mathematical programming where multiple objectives are modeled as a weighted function of the system variables only. The concept of system constraint is retained from constrained optimization formulations. The compromise DSP enhancement has been applied to the robust design of a solar powered irrigation system (Chen et al [5]), vehicle handling performance (Chen et al [6]), jet engine design (Du and Chen [8]) and aircraft design (Simpson et al [28] [19]).

In the compromise DSP, the set of system constraints and bounds define the feasible design space while the set of system goals define the aspiration space. For feasibility the system constraint and bounds must be satisfied. A solution then is that feasible point which achieves the system goals, G_i, as best as possible. Each feasible design point, *Xi,* can be mapped to the aspiration space by achievement functions, $A_j(x)$. The solution to the problem represents a tradeoff between that which is desired (as modeled by the aspiration space) and that which can be achieved (as modeled by the design space). The tradeoffs among the j achievement functions are determined by the deviation function, *Z,* which can be either a weighted sum or preemptive ordering of deviation variables. Correspondingly, the solution is located as the closest feasible design to the aspiration space. In general, the deviation function, *Z,* is minimized to determine the best design point as a compromise among competing achievement functions [19].

Compromise DSP's are written in terms of *n* system variables. The vector of variables, *x,* may include continuous variables and Boolean variables. System variables are independent of the other descriptors and can be changed to alter the state of the system. System variables that define the physical attributes of an artifact must be

positive. A system constraint models a limit that is placed on the design. The set of system constraints must be satisfied for the feasibility of the design. Mathematically, system constraints are functions of system variables only. They are rigid and no violations are allowed. They relate the demand placed on the system, $D(x)$ to the capability of the system, $C(x)$. The set of system constraints may be a mix of linear and nonlinear functions. In engineering problems the system constraints are invariably inequalities. However, occasions requiring equality system constraints may arise. The region of feasibility defined by the system constraints is called the feasible design space.

A set of system goals is used to model the aspiration a designer has for the 0 design. It relates the goal, G_i of the designer to the actual performance, $A_i(X)$ of the system with respect to the goal. The deviation variable is introduced as a measure of achievement because it is desired that the value of $A_i(X)$ equal the value of G_i. Constraining the deviation variables to be non-negative, the system goal becomes:

$$A_i(X) + d_i^- - d_i^+ = G_i \tag{2.1}$$

$$X_j^{\min} \le X_j \le X_j^{\max} \tag{2.2}$$

$$d_i^- * d_i^+ \ge 0 \quad and \quad d_i^- * d_i^+ = 0 \tag{2.3}$$

The product constraint ($d_i^- * d_i^+ = 0$) ensures that at least one of the deviation variables for a particular goal will always be zero.

Bounds are specific limits placed on the magnitude of each of the system and deviation variables. Each variable has associated with it a lower and an upper bound. Bounds are important for modeling real-world problems because they provide a means to include the experience-based judgment of a designer in the mathematical formulation. In

the compromise DSP formulation the aim is to minimize the difference between that which is desired and that which can be achieved. This is done by minimizing the deviation function $Z(d_i^-, d_i^+)$. This function is always written in terms of the deviation variables. All goals may not be equally important to a designer and the formulations are classified as Archimedean or Preemptive --based on the manner in which importance is assigned to satisfying the goals. The general form of the deviation function for m goals in the Archimedean formulation is

$$Z = \sum_{i=1}^{m} Wi(d_i^- + d_i^+), \quad \sum W_i = 1; W_i \geq 0 \tag{2.4}$$

The most general approach for assigning priority is a Preemptive one, in which the goals are rank ordered. This assignment of priority is probably easier in an industrial environment or in the earlier stages of design. Multiple goals can be assigned the same rank or level, in which case, Archimedean styled weights may be used within a level. The measure of achievement is then obtained in terms of the lexicographic minimization of an ordered set of goal deviations. Ranked lexicographically, an attempt is made to achieve a more important goal (or set of goals) before other goals are considered. The mathematical definition of lexicographic minimum (Ignizio [12]) is as follows.

Lexicographical Formulation:

Lexicographical Minimum: Given an ordered array f of nonnegative elements f_ks, the solution given by (I) is preferred to (2) if $f_k^{(1)} < f_k^{(2)}$ and all higher ordered elements (i.e., $f_1, ..., f_{k-1}$) are equal. If no other solution is preferred to f, then f is the lexicographic minimum. Consider, for example, $f_k^{(r)}$ and $f_k^{(s)}$, where $f_k^{(r)}$ = (0, 10, 400, 56) and $f_k^{(s)}$ = (0, 11, 12, 20), then $f_k^{(r)}$ is preferred to $f_k^{(s)}$.

Using the Preemptive formulation, the deviation function is written as:

$$Z = [f_1(d_i^-, d_i^+), ..., f_k(d_i^-, d_i^+)] \tag{2.5}$$

For instance, a problem with four goals may have the deviation function:

$$Z = [f_1(d_1^-, d_2^+), (d_3^-), (d_4^+)] \tag{2.6}$$

The compromise DSP is solved using the Adaptive Linear Programming (ALP) algorithm (Mistree et al. [20]), which is a part of DSIDES (Decision Support in Designing Engineering Systems). The mathematical formulation of the Compromise DSP is as follows.

Find:

The independent system variables

$$X_j \qquad j = 1, ..., n \tag{2.7}$$

And the deviation variables

$$d_i^-, d_i^+ \qquad i = 1, ..., m \tag{2.8}$$

Satisfy:

System constraints (must be satisfied for the solution to be feasible)

$$g_i(X) = 0 \qquad i = 1, ..., p \tag{2.9}$$

$$g_i(X) \geq 0 \qquad i = p+1, ..., p+q \tag{2.10}$$

System goals

$$A_i(X) + d_i^- - d_i^+ = G_i \tag{2.11}$$

Bounds

$$X_j^{\min} \leq X_j \leq X_j^{\max} \tag{2.12}$$

$$d_i^- * d_i^+ \geq 0 \quad and \quad d_i^- * d_i^+ = 0 \tag{2.13}$$

Minimize

Case a: Pre-emptive (lexicographic minimum)

$$Z = [f_1(d_i^-, d_i^+), ..., f_k(d_i^-, d_i^+)] \tag{2.14}$$

Case b: Archimedean

$$Z = \sum_{i=1}^{m} Wi(d_i^- + d_i^+), \quad \sum W_i = 1; W_i \geq 0 \tag{2.15}$$

2.4 Design of Engineering Systems (DSIDES)

Decision Support Problems (DSPs) provide methods to solve optimization problems in design, manufacturing and maintenance. For real world problems in early stages of design, the data available is not so accurate, and hence, optimization cannot reach the desired goals. DSP achieves the result by continuously improving the initial solution. DSP can be used to solve a variety of decision making problems like the Selection, Compromise, Hierarchical, and Conditional. Compromise DSP is a hybrid formulation of mathematical programming and goal programming. There are many methods for solving the compromise decision support problems. Adaptive Linear Programming (ALP) is one such method implemented in software called Design of Engineering Systems (DSIDES). A further extension to the compromise DSP approach is the Multidisciplinary Robust Design Optimization. This is a combination of the bi-level collaborative optimization approach and the Compromise Decision Support Problem [20].

2.5 Parallel Computation

Parallelism is a strategy for performing large, complex tasks faster. A large task can either be performed serially, one step following another, or can be decomposed into

smaller tasks to be performed simultaneously, i.e., in parallel. Parallelism can be achieved by dividing the task into smaller tasks, assigning the smaller tasks to multiple workers to work on simultaneously, and coordinating the workers. Parallel problem solving is very common e.g., building construction, operating a large organization, automobile manufacturing plant etc. The automobile example is especially relevant for IE applications.

2.5.1 Application of Parallel Computation

With the increased reliance on technology to solve the problems of modern age there are several classes of problems that require faster processing. Broad Categories of problems include Simulation and Modeling problems, Problems dependent on computations or manipulations of large amounts of data, and Grand Challenge Problems. Simulation and Modeling problems include problems based on successive approximations and problems requiring more calculations with more precision. Problems dependent on computations or manipulations of large amounts of data include Image and Signal Processing, Entertainment (Image Rendering), Database and Data Mining, and Seismic studies.

Grand Challenge Problems are defined as “fundamental problem in science and engineering with broad economic and scientific impact, whose solutions can be advanced by applying high performance computing techniques and resources." Some common example of the Grand Challenge Problem are Climate Modeling, Fluid Turbulence, Pollution Dispersion, Human Genome, Ocean Circulation, Quantum Chromo-dynamics, Semiconductor Modeling, Superconductor Modeling, Combustion Systems, and Vision & Cognition sciences.

2.5.2 Benefits of Parallel Computation

With the use of these powerful parallel computers, engineers and scientists can design products such as airplanes, cars, electronic components, and pharmaceuticals. Parallel computers are also used to improve processes like oil reservoir management, toxic waste cleanup, airline scheduling, mutual fund management, and video-on-demand. Scientists are probing important problems in chemistry, biology, geology, astronomy, and physics through detailed models generated by parallel computers. Parallel computation provides the ability to achieve performance and investigate problems impossible with traditional computers. It exploits the processors, memory, disks and tape system and provides the ability to scale to problem. The ability to quickly integrate new elements into systems and the low cost associated with distributed memory parallel clusters are also greatly advantageous.

2.5.3 Limitations of Parallel Computation

Like all technological issues parallel computation has its own limitation. There are many new approaches for parallel programming which are still in the development stages. High Performance Fortran and LISP are some of the languages being developed. Programmers need to learn parallel programming approaches. Standard sequential codes will require modifications and some of the legacy code may have to rewritten from the scratch. Compilers and tools are often not mature and still in development stages. There is also a lack of standardization. In many areas of scientific computing it is necessary to assure seamless cooperation between multiple software components like grid generation, adaptive grid refinement and problem partitioning. Novel approaches to parallel computing are required to tackle these issues. I/O handling is not very well understood

yet and most of the programs have a sequential I/O handling component. Thus full advantage of parallelization cannot be achieved for I/O intensive programs.

2.5.4 Elements of Parallel Computation

A working cluster of a parallel computer requires many elements. The main elements are multiple processors, Network, Environment to create and manage parallel processing, and a parallel algorithm and a parallel program. Multiple processors can be considered as multiple workers employed to do a single job by sharing the work among them. The Network is the medium or link between the workers through which the work related information and data can be shared.

The environment to create and manage parallel processing consists of an Operating System and Parallel Programming Paradigms. Operating system in the sense of manufacturing is like a production manager who knows how to handle multiple workers. Parallel Programming Paradigms are a set of schemes to distribute the work among the different processors. Message Passing and Data Parallel are some of the more popular paradigms. There are other paradigms as well and OpenMP, Shmen are some of them.

Message Passing includes Message Passing Interface (MPI) and Parallel Virtual Machine (PVM). FORTRAN 90 / High Performance FORTRAN are extensively used for data parallel implementation. A parallel algorithm and a parallel program decompose the problem into pieces that multiple workers can perform.

Parallel programming involves the decomposition of an algorithm or data into parts, distributing the parts as tasks which are analyzed by multiple processors simultaneously and coordinating the work and communications of those processors.

Types of parallel architecture and the type of processor communication used are the two main considerations while developing a parallel program.

2.5.5 Parallel Architecture

All parallel computers use multiple processors, and there are several different methods used to classify computers. Because of the diversity of the problems there is not a single taxonomy fits all designs. The original classification of parallel computers is known as Flynn's taxonomy. Flynn's taxonomy uses the relationship of program instructions to program data. Flynn classified machines in four categories according to the number of instructions and the number of data streams [21].

The four categories are:

- SISD - Single Instruction, Single Data Stream
- SIMD - Single Instruction, Multiple Data Stream
- MISD - Multiple Instruction, Single Data Stream
- MIMD - Multiple Instruction, Multiple Data Stream

On one extreme is the single-instruction single-data (SISD) and on the other extreme is the Multiple-instruction multiple-machine (MIMD) system. The key difference between SISD and MIMD systems is that with MIMD systems, the processors are autonomous. Each processor is capable of executing its own program. MIMD systems are divided into shared memory and distributed memory systems. These two systems can also be visualized as multiprocessor and multi computers. Figure 2.4 depicts a generic shared memory parallel cluster and Figure 2.5 depicts a generic distributed memory cluster.

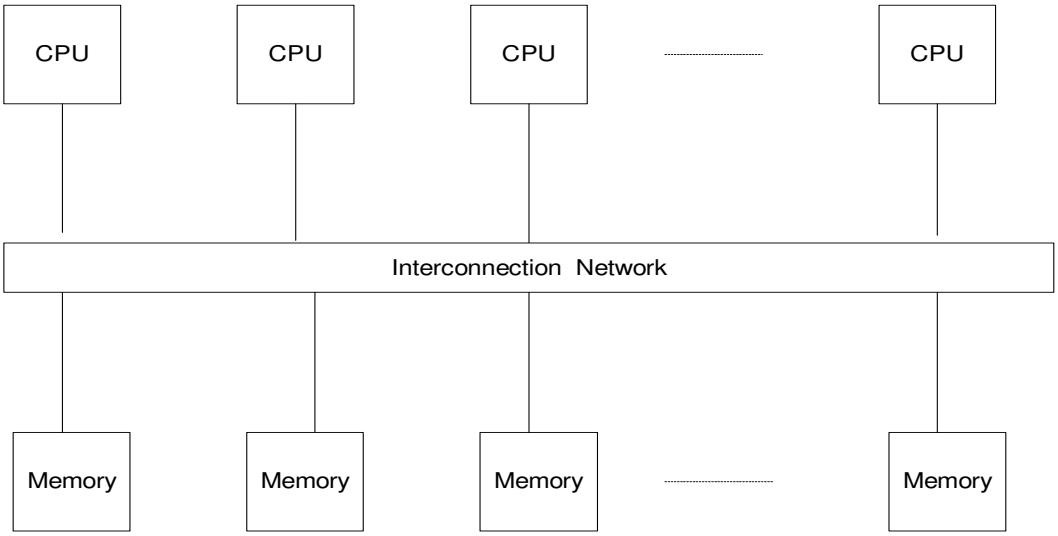

Figure 2.4: Generic Shared Memory Architecture

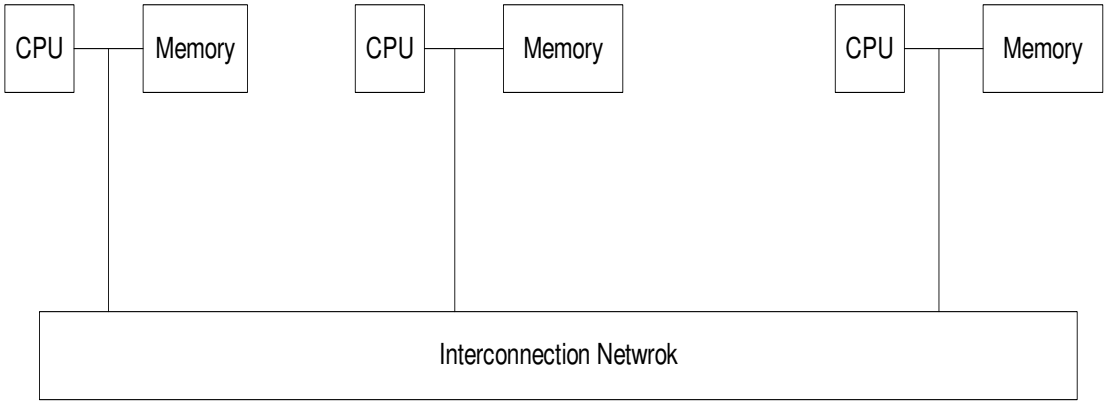

Figure 2.5: Generic Distributed Memory system

2.5.6 Speedup and Scalability

Amdahl's Law states that potential program speedup is defined by the fraction of code (P) which can be parallelized:

$$speedup = \frac{1}{1-P}$$

where

P = fraction of parallel code

If none of the code can be parallelized, P = 0 and the speedup = 1 (no speedup). If all of the code is parallelized, P = 1 and the speedup is infinite (in theory). If 50% of the code can be parallelized, maximum speedup = 2, meaning the code will run twice as fast. Introducing the number of processors performing the parallel fraction of work, the relationship can be modeled by:

$$speedup = \frac{1}{\frac{P}{N} + S}$$

where

P = fraction of parallel code

N = Number of processors

S = Fraction of serial code

Table 2.2 Amdahl's Law

N	P = 0.50	P = 0.90	P = 0.99
10	1.82	5.26	9.17
100	1.98	9.17	50.25
1000	1.99	9.91	90.99
10000	1.99	9.91	99.02

It soon becomes obvious that there are limits to the scalability of parallelism. For example, at P = .50, .90 and .99 (50%, 90% and 99% of the code is parallelizable). One of the main bottlenecks to the parallel computing is the transfer of data between memory

and CPU. Shared memory systems fare better in terms of performance, but they are very expensive. Distributed memory systems are less costly, but there is a large communication overhead attached with these systems. Most of the supercomputers in educational institution including LSU's SuperMike are distributed memory systems.

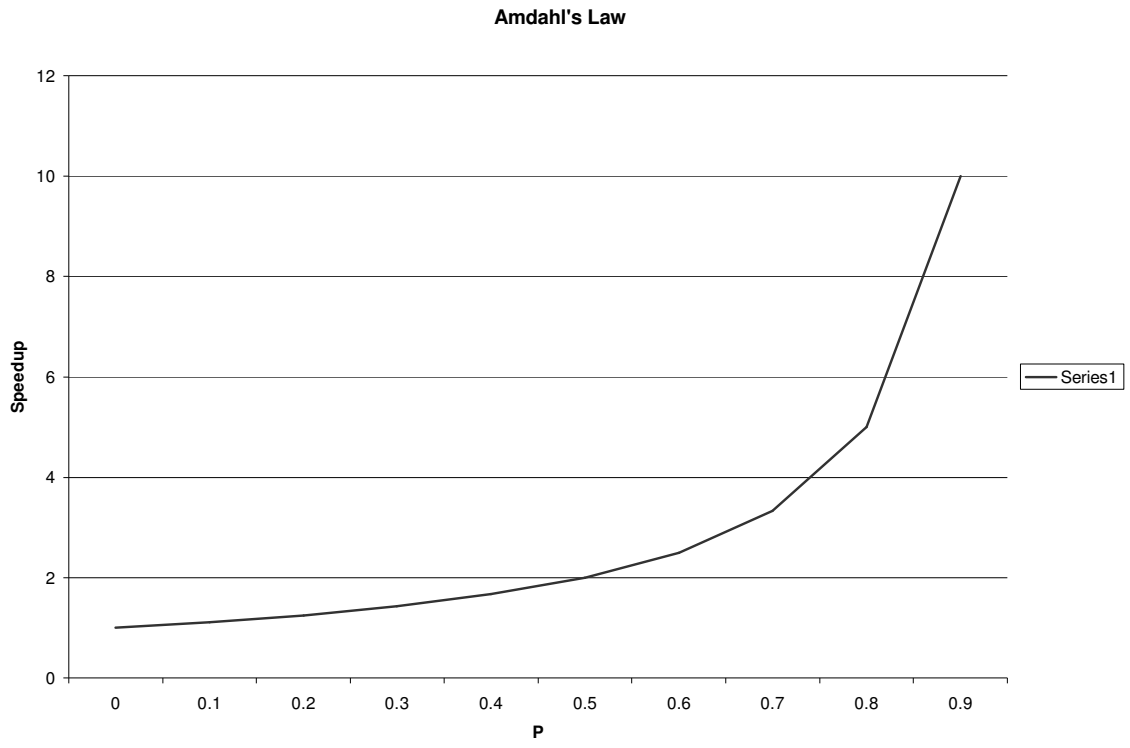

Figure 2.6 Amdahl's law

One of the main challenges to the successful implementation of MDO has been the very high values of solution times. In certain cases it takes years of computer time to solve a particular problem. With the advent of parallel computers it has become possible

to limit the solution times to reasonable levels. In fact some of the problems which were earlier thought as non-solvable can now be solved.

There are many languages used in parallel programming but the most popular of them all for distributed environment is the message passing interface (MPI) which works with C, C++ and FORTRAN. MPI provides a set of libraries which help communicate between different processors.

3. Literature Review

This literature survey for this research can be classified in three broad categories.

- Multidisciplinary Design Optimization (MDO)
- Design of Engineering Systems (DSIDES).
- Parallel Computation

They are linked to each other as follows: Problems are formulated as an MDO problem which can be solved using DSIDES in a parallel computing environment. Many researchers are working on the parallel implementation of different MDO frameworks, but most of the research is being done by industry especially the aircraft industry and government agencies like NASA. Simpson [27] discusses the research efforts related to the broad areas of MDO at various universities, government organizations and Industries. SGI and Ford integrated parallel programming and response surface models for rapid visualization of design alternatives to enable design steering during MDO. Penn State is collaborating with Boeing and Lockheed Martin Space Systems to develop visualization interfaces to support design decision-making. University of Clemson also reports studying two aspects of collaborative design: managing information from large, distributed optimizations, and using design exemplars to capture, retrieve, and manipulate knowledge [27].

University at Buffalo, RPI, MIT, Notre Dame, Arizona State, Northwestern, Wright State, University of Michigan, and Georgia Tech are some of the other universities where research with respect to various stages of MDO is being conducted. Sandia National Labs, Vanderplaats R&D, NASA and Boeing are some organizations apart from SGI and Ford where extensive MDO based research is being carried out [27].

Kodiyalam et al. [14] have discussed the various MDO frameworks. They have also listed some of the requirements for framework. The application of different framework and their appropriateness to a viable vehicle design has been discussed.

Amitay et al. [1] have designed the framework to integrate the disciplinary analyses of distributed over intranet. They have extended the basic concepts of a distributed environment to a heterogeneous distributed computing environment. Another important area is wrapping analysis modules for integration with the framework. Hamdi et al. [10] have assessed the various strengths and limitations of a cluster of workstations by capturing the effects of the above issues by evaluating the performance of this computing environment in the execution of a parallel ray tracing application through analytical modeling and extensive experimentation.

Kodiyalam et al. [15] have investigated some alternate sampling and meta-modeling methods for MDO solution of realistic aerospace design problems in a multi-processor, high performance computing environment. The methods investigated in this work for MDO solution are comparatively simpler than the existing, formal MDO approaches. Based on the trends in massively parallel processing and HPC (High Performance Computing), it is expected that the MDO methods will become simpler as well as easier to understand and use with complex design problems. The conceptual simplicity of the MDO approach is some times lost because of the large computing labor in the sampling. That labor is effectively compressed in time by the HPC environment that operates a large number of processors concurrently.

Wujek et al. [32] have reviewed the recent implementation advances and modifications in the continued development of a Concurrent Subspace Optimization (CSSO) algorithm for Multidisciplinary Design Optimization (MDO). The CSSO-MDO algorithm implemented in this research incorporates a Coordination Procedure of System Approximation (CP-SA) for design updates. Their study also details the use of a new discipline-based decomposition strategy which provides for design variable sharing across discipline design regimes (i.e., subspaces). The algorithm is implemented in a distributed computing environment using the graphical user interface, providing for truly concurrent discipline design. They have reported significant time savings when using distributed computing for concurrent design across disciplines. The use of design variable sharing across disciplines does not introduce any difficulties in implementation as the design update in the CSSO-MDO algorithm is generated in the CP-SA. Application of the CSSO algorithm results in a considerable decrease in the number of system analyses required for optimization in both test problems. More importantly, for the fully coupled aircraft concept sizing problem, a significant reduction in the number of individual contributing analyses is observed.

Manolache et al. [17] have discussed the various opportunities of parallel processing (PP) at four algorithmic levels of the approaches, i.e., the subsystem solver level, the subsystem optimization level, the full system optimization level, and the sequence of problems level. Advantages of different PP implementations of the MDO approaches are outlined. Special emphasis is put on vertical PP processing, where one thread treats a hierarchical structure (e.g., a full system evaluation), inter-thread communication is low, and processor loads are uniform.

Becker at al. [2] have demonstrated that the programming language Java offers substantial possibilities for the type of complex engineering problems typically encountered in Multidisciplinary Design Optimization (MDO) problems. They have developed a web based application in which one computer is designated as the server and sends out required inputs to a number of client subsystems over the Internet. A number of client computers can connect to the server and then receive the inputs necessary to calculate the solution to their model. As the code necessary to solve the model already exists at the client, only the inputs need to be sent over the network. When the client has solved the calculation, it returns the results to the server which processes the result to produce new inputs. Results of a number of parametric studies on the behavior of complex systems in a distributed environment have also been reported in this paper.

4. Methodology

This chapter discusses the methodology of this research. It discusses the program architecture, code modifications, metrics chosen and the methodology and motivation behind benchmarking studies.

4.1 Program Architecture

DSIDES is composed of four main program components

- *alpctl.f90*
- DSIDESLIB.a
- User provided files
- Scripts

Alpclt.f90 is the main program which uses the subroutines provided by the user and the library of subroutines *DSIDESLIB.a* to run the optimization. Users must provide two files with the same name but extensions *.f90* and *.dat* to perform the optimization. The script *runalp* is used to run the program. Script uses *Makefile* to compile the user supplied subroutine during the runtime. Use of *Makefile* saves a lot of compilation effort because only the files which have been modified are recompiled.

4.2 Sequential Program Flow

All optimization processes are conducted sequentially. Both the subsystem codes wait while the system level optimization is taking place. Similarly, the system and one of the subsystems wait while the other subsystem is being optimized. Figure 4.2 illustrates the program flow for sequential execution.

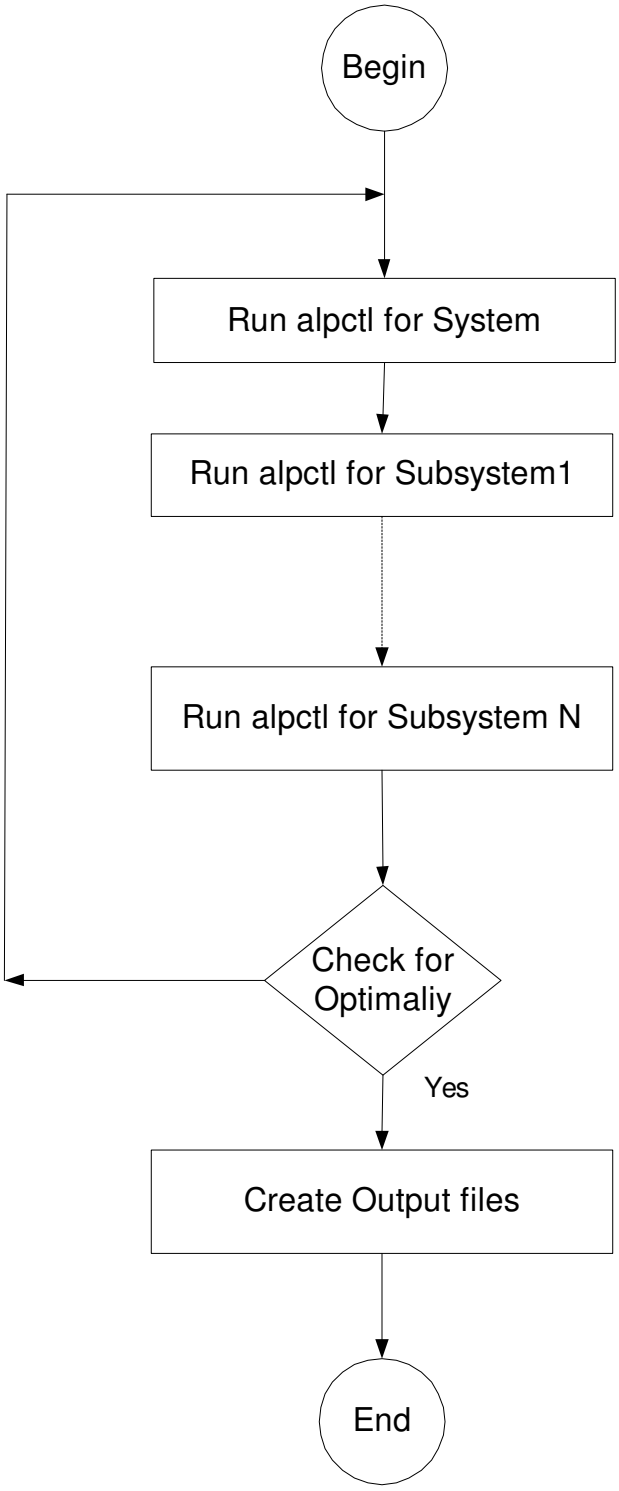

Figure 4.2: Sequential Program Flow

4.3 Proposed Program Architecture

In the proposed parallel architecture, rather than using the power one processor, both the subsystem level optimization is simultaneously carried out. This is possible because the problem is formulated in such a way that there are no coupled variables.

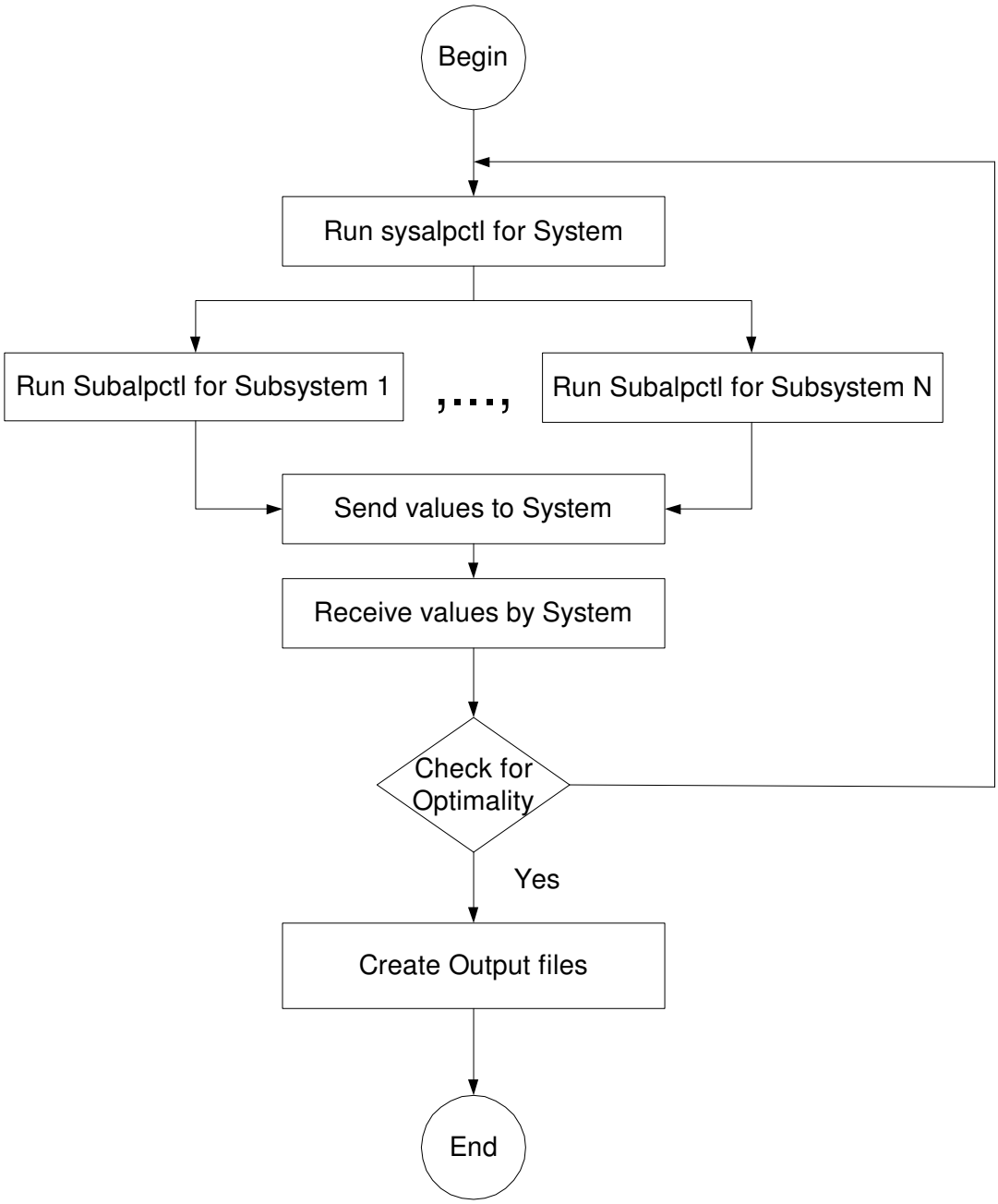

Figure 4.1 Parallel Program Flow

Though there is significant communication overheard, parallel system is expected to run faster than the sequential case. As the size and complexity of the problem increases it is expected that the parallel model will run significantly faster.

4.4 Algorithm and Code Modifications

In the sequential program when the system optimizer finishes its work it calls the subsystem optimizer sequentially on after another. Since both the subsystems did not have coupled variables, it provided an opportunity for parallelization. Rather than one processor handling all the subsystem calls, the code was modified so that each subsystem can be handled by a separate processor.

A new subroutine, sysmain (Appendix A.1) was added and to send and receive the data from each subsystem. This system was called from the same point where the initial calls to sub system optimizers were made as depicted in the section of code below.

```
!!!! Sequential program Combustsys.f90
     call runalpgeo(b,dI,dE,bgeo,dIgeo,dEgeo,dgeo)
     call runalpdyn(b,dI,cr,w,bdyn,dIdyn,crdyn,wdyn,ddyn)
!!!
!!! Parallel Program Combustsys.f90
CALL
sysmain(b,dI,dE,bgeo,dIgeo,dEgeo,dgeo,cr,w,bdyn,dIdyn,crdyn,wdyn,ddyn,flagsys)
!!!
```

Also there were some issues with file handling. DSIDES as discussed earlier was designed to run in a sequential mode; hence, the files and their unit numbers were not assigned as run time. In parallel execution this posed the problem of different processor

opening the same file simultaneously, which cannot be done. There was also problem of multiple processors writing to the same output file, thereby corrupting it.

These issues were handled by making separate alpctl.f90 files for each subsystem and the system. These files were named as *sub1alpctl.f90, sub2alpctl.f90* and *sysalpctl.f90* respectively. Within these files names and unit numbers of the input and output files were changed. For the sake of convenience all unit numbers were hard coded, but there are libraries available to dynamically assign the unit numbers and should be used for more complex implementations. Additional code was added to the *sysalpctl.f90* file to send and receive data as well perform some basic parallelization functions like initialization and finalization.

Separate scripts were created to run the system and each. This was necessary because the original input files are moved to the template file *ALPINP.dat* in the script. Similarly the generic output file *ALPOUT.dat* is moved to the specific output file by the script. These generic names were changed to *ALPINP1.dat* and *ALPINP2.dat* for the subsystems where as *ALPINP.dat* was used for the system. Same naming convention was used for the *ALPOUT.dat* files as well.

4.5 Benchmarking Studies

The full scope of the benefits of a parallel approach cannot be measured by one problem. On the same note there might be issues which are not captured due to the one problem. To study a large array of problems, benchmarking codes were developed for sequential [Appendix A.2] as well as parallel approaches [Appendix A.3].

These programs were developed to simulate the optimization process in *DSIDES.* Number of Subsystems, Number of Analysis Cycle, Number of Synthesis Cycle, and

Time for one Analysis Cycle and Time for one Synthesis Cycle were kept as variables. These variables were read via an input file and by changing these parameters different aspects of parallelization were studied.

4.6 Performance Metrics

Savings was adopted as the generic indicator for improvement. It can be defined as the percentage change in the wall time between sequential and parallel runs.

Mathematically it can be represented as follows,

$$I = \left(\frac{t_{seq} - t_{par}}{t_{seq}} \right) \times 100$$

(4.1)

Where,

I = Savings

t_{seq} = Wall Time for sequential run

t_{par} = Wall Time for parallel run

One other metrics which has been used is Efficiency [9]. This has been used in benchmarking studies to measure performance improvements for runs with different number of subsystem, i.e. different number of processors used.

Mathematically this can be defined as follows,

$$I_p = \frac{t_{seq}}{t_{par} \times n} \qquad (4.2)$$

Where,

I_p = Efficiency

t_{seq} = Simulation time for sequential run

t_{par} = Simulation time for parallel run

n = number of subsystem

Although the total number of processors is one more than the number of subsystems, in this formula n is equal to the number of subsystems because at least one processor is idle all the time during program execution. Thus, a value of n equal to the number of sub systems better reflects the actual improvement obtained.

5. Results and Conclusion

5.1 Results

The results of various experiments are presented in this section. This section starts with the system description and then the results of various experiments are discussed. Results have been tabulated and supplemented with graphs as necessary for better presentation.

5.1.1 System Description

All experiments were done in three modes, Sequential, Multithreading and the actual distributed parallel computer cluster. The distributed cluster consists of 32 slave nodes and one master node. Each slave node is powered by Pentium processors. The detailed system description for the slave node is illustrated in Table 5.1.

Table 5.1: System description-Slave Nodes

Processor	Pentium II 266 MHz
No of processor/Machine	1
Memory	256 Mega Bytes (MB)
Operating System	Red Hat Linux
Operating system version	7.0
FORTRAN Compiler	Intel FORTRAN (IFC)
FORTRAN Compiler version	8.0
MPI	MPICH 1.2.5
Network Connection	10/100 Ethernet

Table 5.2: System description-Master Nodes

Processor	Athlon 2400
No of processor/Machine	2
Memory	2 Giga Bytes (GB)
Operating System	Red Hat Linux
Operating system version	7.0
FORTRAN Compiler	Intel FORTRAN (IFORT)
FORTRAN Compiler version	8.0
MPI	MPICH 1.2.5
Network Connection	10/100 Ethernet

The master node is powered by Athlon processors. The detailed system description for the master node is illustrated in Table 5.2.

5.1.2 Results in Multithreading Mode

The complete program was run in multithreading mode. Maximum number of of Analysis cycle and synthesis cycle were 300 and 100 respectively. This program was run on the master node which is a very powerful machine, and hence the whole computation could be finished in hours as compared to days in the distributed environment.

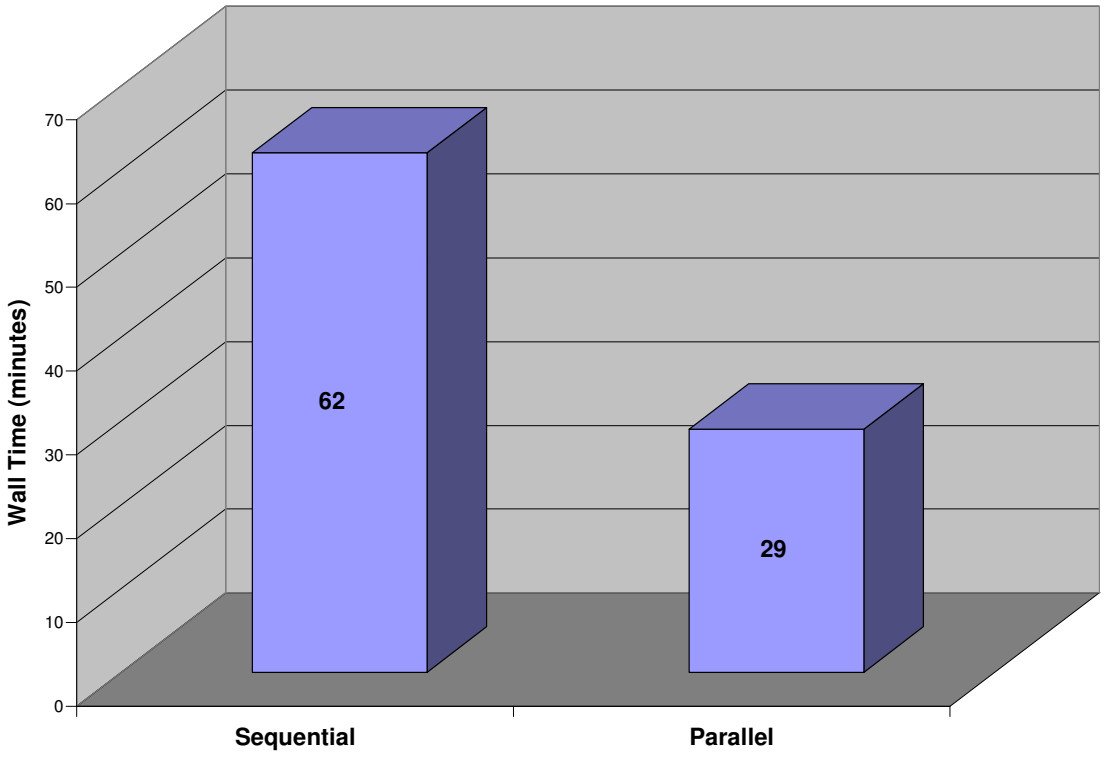

Figure 5.1: Wall Time for multithreading mode

For the case of multithreading implementation speed up is found to be 53.2%. This result is very close the theoretical expectation. In a multithreading implementation

data is transferred on the system bus rather than the network. Since the system bus is much faster than an Ethernet network the downside of communication overhead are minimized.

Table 5.3: Multithreading results

Program name	**Run type**	**Wall Time (Minutes)**	**Savings (%)**
Combustsys	Sequential	62	53.2
Combustsys	Parallel	29	

This result also illustrates an interesting opportunity. Even without having access to a cluster of computers, the total solution times can be decreased. The idea here is to use the power of a single processor more efficiently. For implementations of mid-sized problems which are not very resource intensive (Memory and CPU time), this can be a good strategy to decrease simulation times.

5.1.3 Results in Distributed Environment

The combustion chamber as described in chapter 1 was run in the distributed computing environment. Number of Analysis cycle which controls the simulation time for the problem is varied in ratio of two starting with the initial value 2. Experiments were run for 2, 4, 8, 16, and 32 Analysis cycles. The main idea behind varying the number of Analysis cycle was to study the impact of length of simulation on savings. Results of various simulation runs are tabulated below. Wall Times for parallel and sequential runs for a given number of Analysis cycle have been graphically compared.

Table 5.4: Simulation results for no of Analysis Cycle = 2

	b	***dI***	***dE***	***cr***	***w***	***Objective Function***	**Wall Time (Minutes)**	**Savings (%)**
Sequential	78.90	34.95	29.54	8.28	10.52	4.58	78:41	25.5
Parallel	78.90	34.92	29.40	8.28	10.52	4.58	58:47	

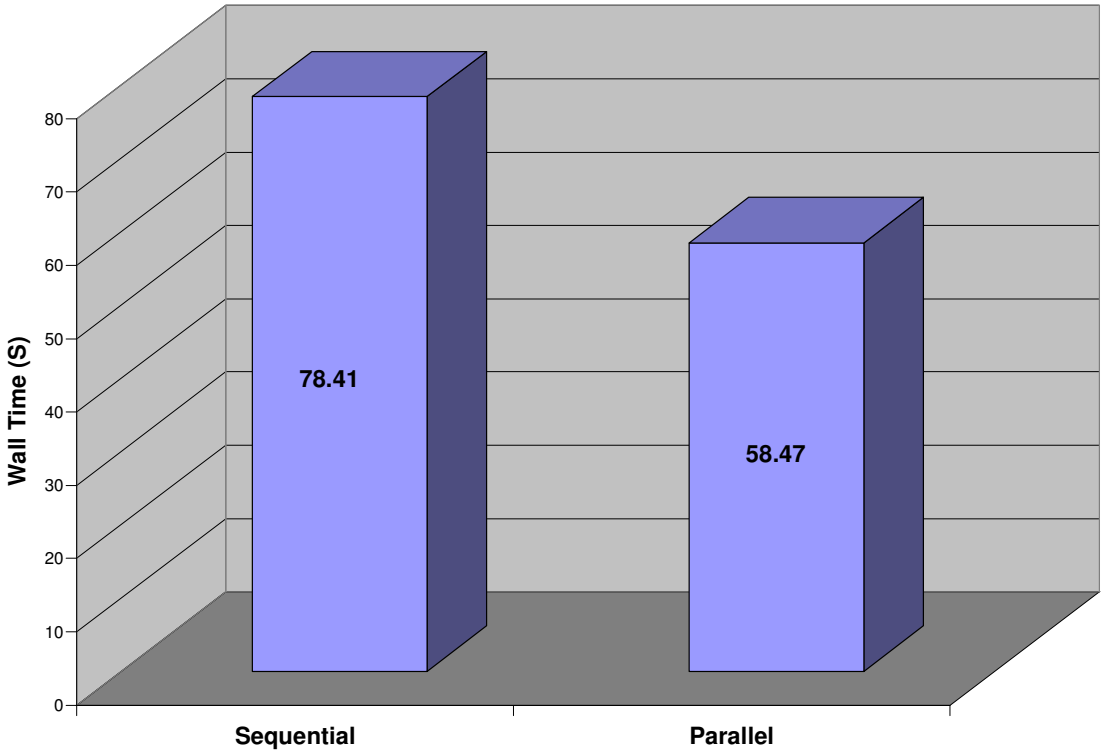

Figure 5.2: Total Wall Times for Analysis Cycle = 2

Table 5.5: Simulation results for number of Analysis Cycle = 4

	b	*dI*	*dE*	*cr*	*w*	*Objective Function*	**Wall Time (Minutes)**	**Savings (%)**
Sequential	79.03	35.07	29.76	7.14	7.76	0.885659	149	23.5
Parallel	78.90	35.28	29.23	7.14	7.76	0.883883	114	

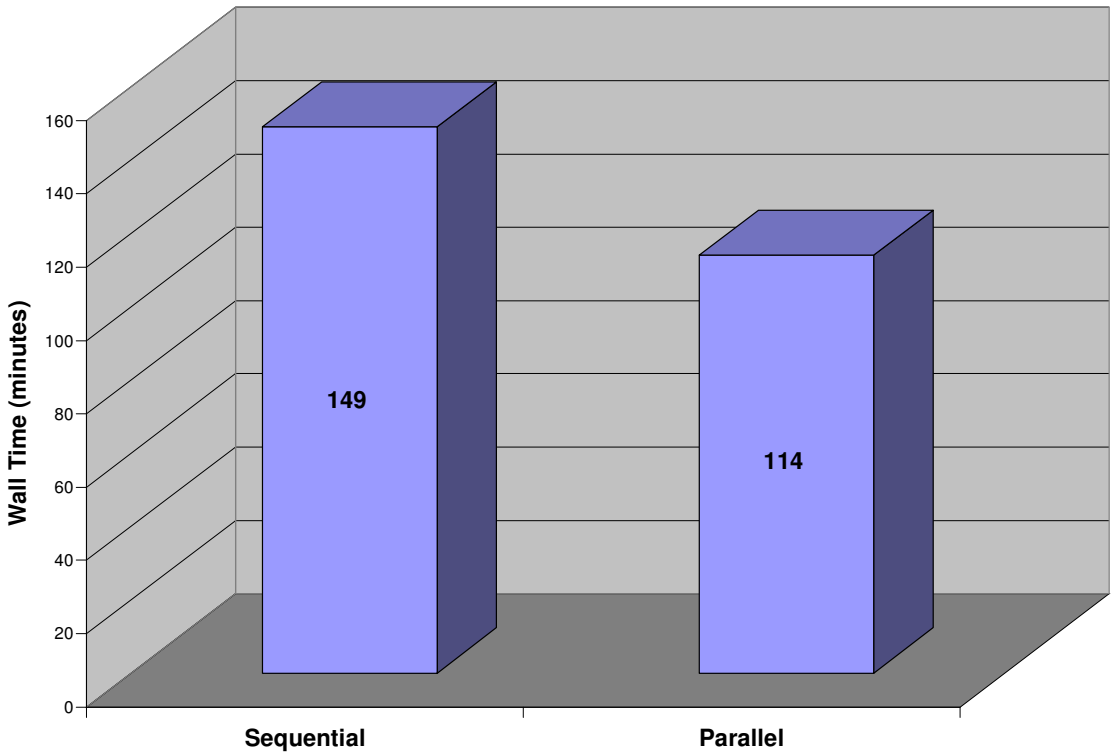

Figure 5.3: Wall Time for Analysis Cycle = 4

Table 5.6: Simulation results for number of Analysis Cycle = 8

	b	***dI***	***dE***	***cr***	***w***	***Objective Function***	**Wall Time (Minutes)**	**Savings (%)**
Sequential	79.02	35.07	29.76	7.14	7.76	0.885659	292	25
Parallel	78.90	35.28	29.23	7.14	7.76	0.883883	219	

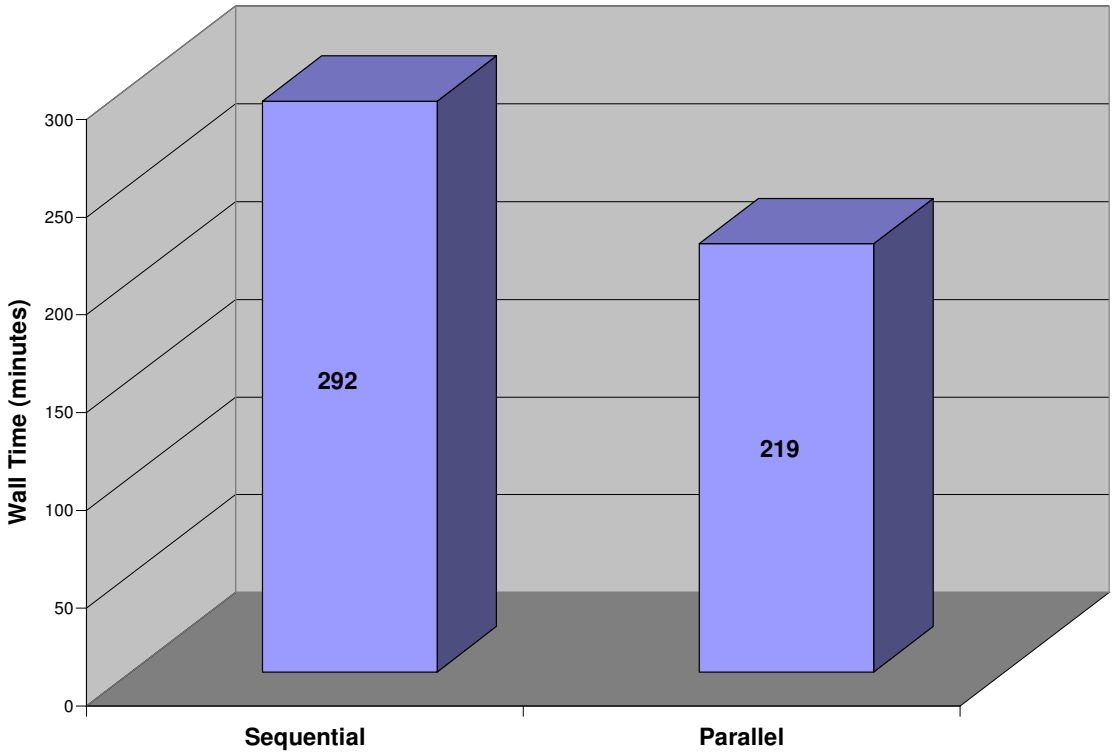

Figure 5.4: Wall Time for Analysis Cycle = 8

Table 5.7: Simulation results for number of Analysis Cycle = 16

	b	***dI***	***dE***	***cr***	***w***	***Objective Function***	**Wall Time (Minutes)**	**Savings (%)**
Sequential	79.02	35.07	29.76	7.14	7.76	0.885659	562	22.95
Parallel	78.90	35.28	29.23	7.14	7.76	0.883883	433	

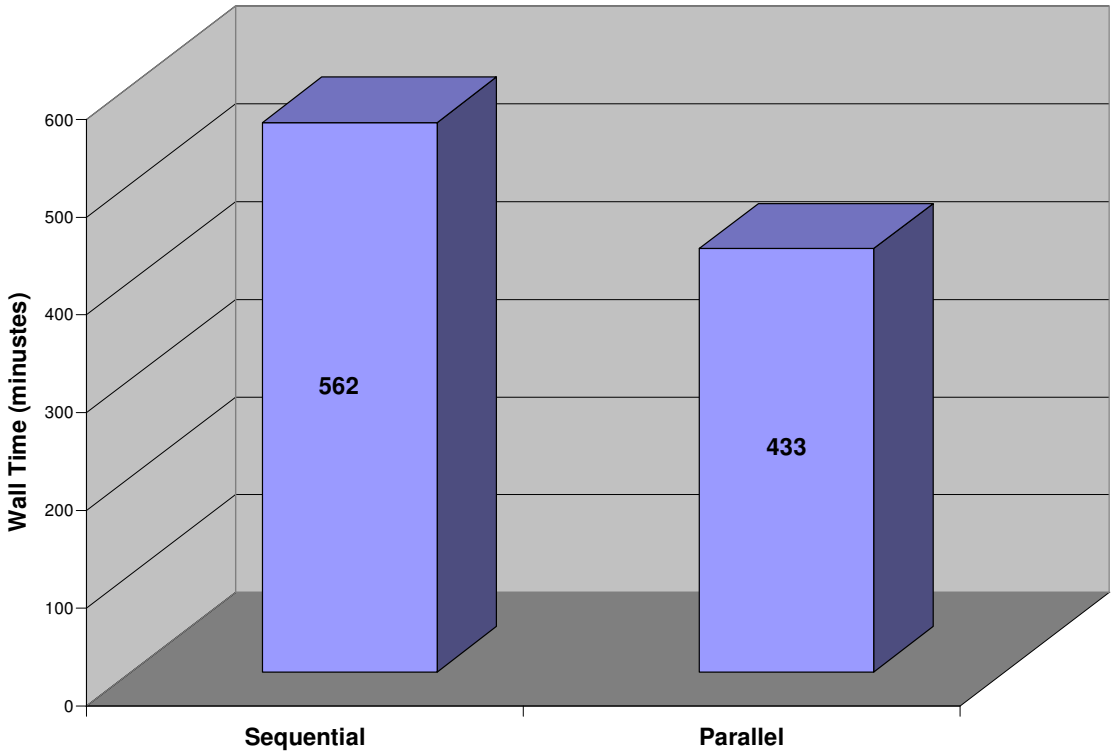

Figure 5.5: Wall Time for Analysis Cycle = 16

Table 5.9: Simulation results for number of Analysis Cycle = 32

	b	***dI***	***dE***	***cr***	***w***	***Objective Function***	**Wall Time (Hours)**	**Savings (%)**
Sequential	77.37	29.06	26.77	6.07	7.03	0.893510	19.3	28.6
Parallel	78.90	35.28	29.23	7.14	7.76	0.883883	13.8	

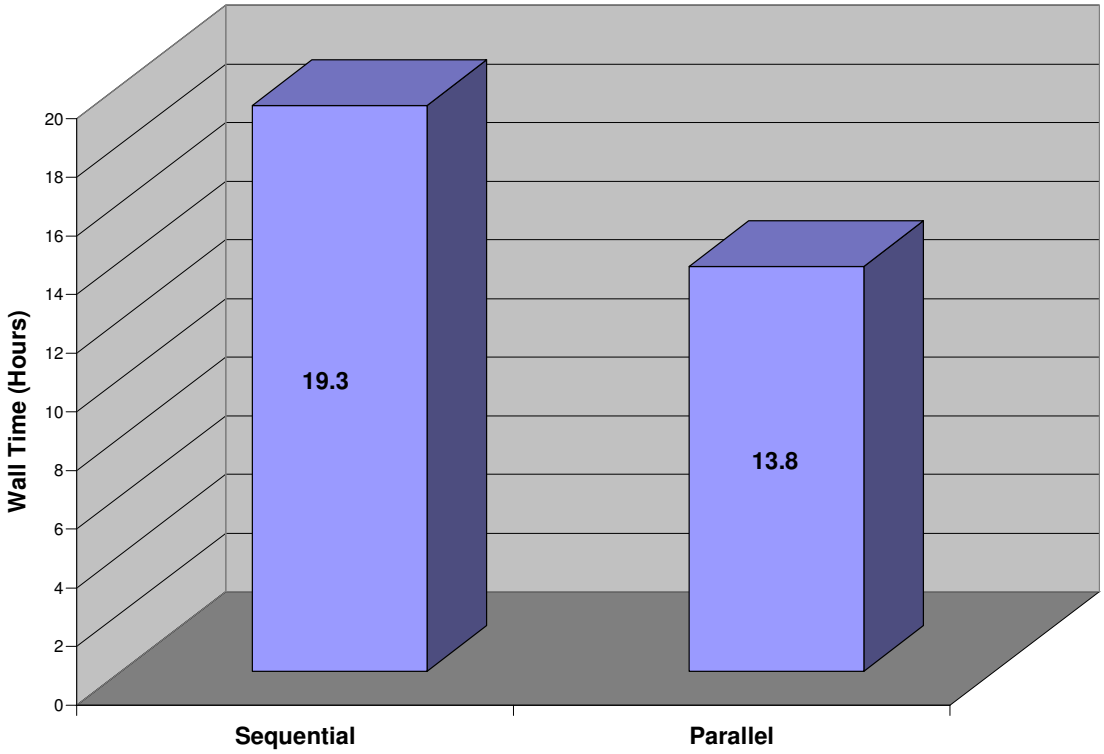

Figure 5.6: Wall Time for Analysis Cycle = 32

Savings (%) as a function of number of Analysis cycle has been plotted in Figure 5.7 and Figure 5.8. As both the Figures indicate there is no definite trend in the Savings (%). The values alternate between increase and decrease which increase in the number of Analysis cycle.

Wall Time in the parallel framework has decreased around 25% in all the runs. Theoretically, if three processors are used then the Wall Time should have decreased by 67%. This relatively low speedup for this implementation can be explained in two ways:

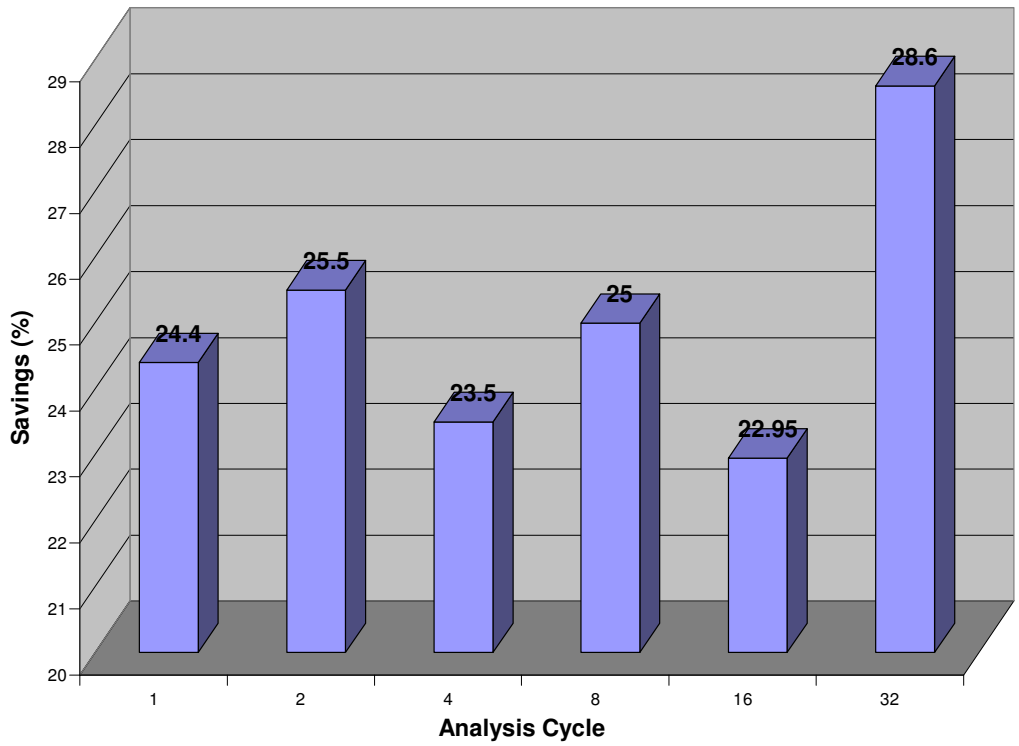

Figure 5.7: Comparison of Savings vs. Number of Analysis Cycle

1. Though three processors are being used but at any given time only two processors are active. Once the processor handling the system sends it value it remains idle, and only the two processors running the subsystems are active.

2. The Wall Time for a single iteration of the geometry and thermodynamics subsystem are 140 seconds and 38 seconds respectively. Since the Wall Times are not the same,

hence one processor must wait for the other process to finish execution before it can start its next iteration. This further explains the below the expected decrease in Wall Time.

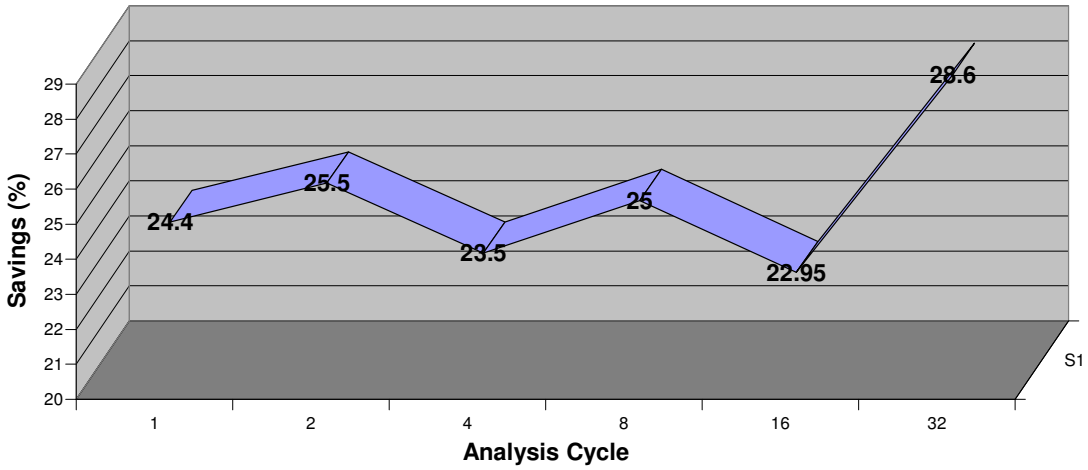

Figure 5.8: Comparison of Savings (%) vs. Number of Analysis Cycle

For different set of Analysis cycles, the Savings has a mean of 25% and variance of 4%. The variations are random and do not seem to be correlated either negatively or positively with the number of Analysis cycle in any way.

5.1.4 Results for Benchmarking Studies

Benchmarking studies were carried out to study a larger set of problems. Three set of experiments with number of subsystem equal to 4, 8, and 16 were carried out. For each set of experiments three separate runs were made for Wall Time (Synthesis and Analysis)

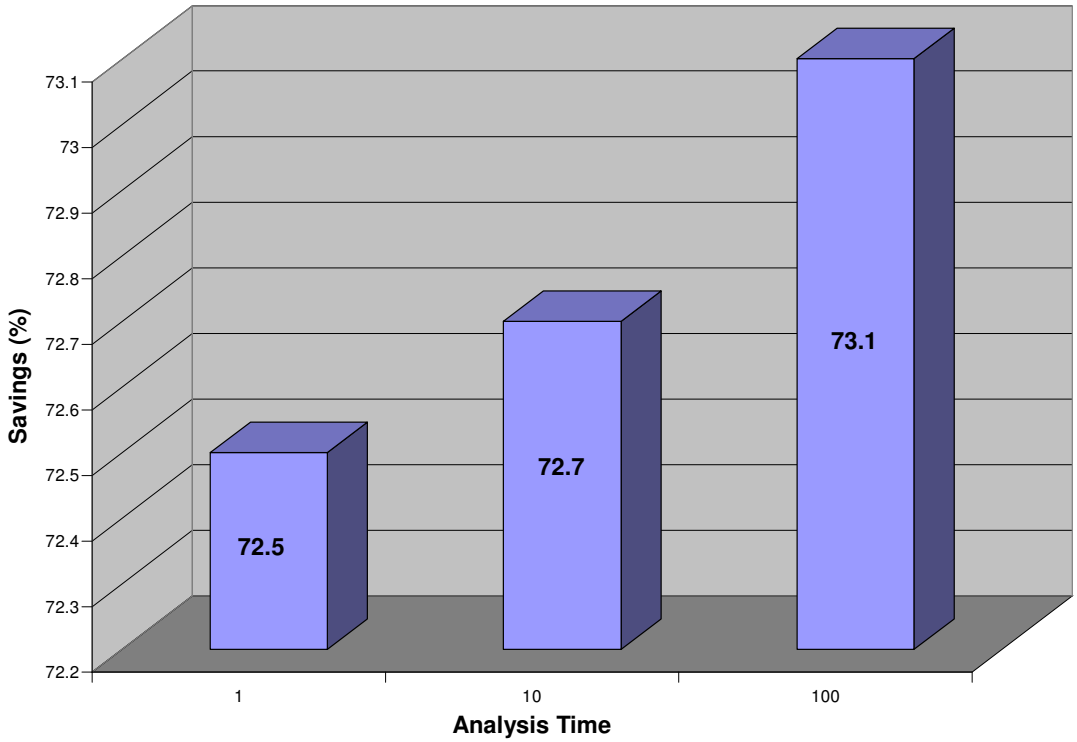

Figure 5.9: Savings (%) for N=4

Table 5.8: Simulation results for N=4

Run Type	Time	Wall Time (Seconds)	Efficiency	Savings (%)
Sequential	1	1232	0.91	72.5
Parallel	1	339		
Sequential	10	12299	0.92	72.7
Parallel	10	3360		
Sequential	100	122972	0.93	73.1
Parallel	100	33053		

being equal to 1, 10, and 100 seconds respectively. These experiments were run using both sequential and parallel benchmarking code. Number of Analysis and Synthesis Cycle was kept constant 30 and 10 respectively for all runs.

It can be seen in Table 5.8 that Savings (%) is positively correlated with the Analysis time. In other words as the complexity of the problems increases, larger savings are obtained. A higher savings (%) indicates a larger decrease in the total Wall Time.

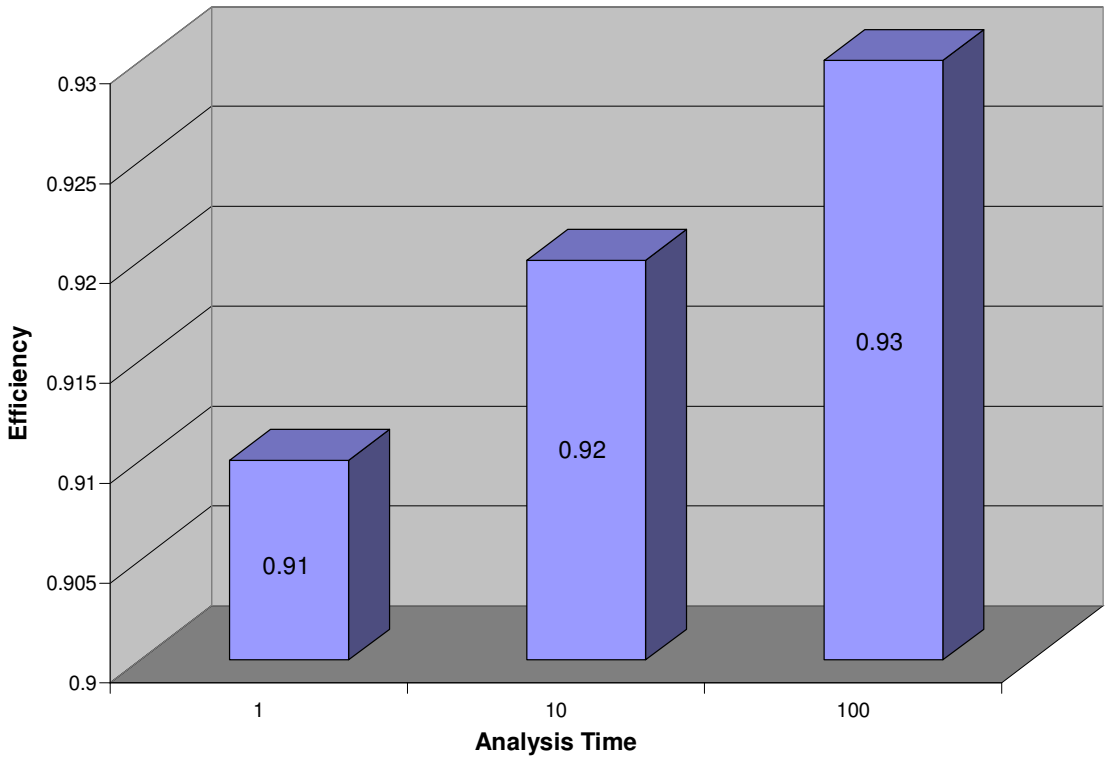

Figure 5.10: Efficiency for N=4

The total Wall Time is lower for the parallel execution as compared to the sequential execution. This is the expected result because more computing power is being used hence total Wall Time should decrease. Similarly, Efficiency is also positively correlated with the Analysis time. A value of 1 will indicate that there was no communication overhead whereas a value of 0 will indicate that there is an no processing, only communication.

Table 5.9: Simulation results for N=8

Run Type	Time	Wall Time (Seconds)	Efficiency	Savings (%)
Sequential	1	2435	0.69	81.9
Parallel	1	441		
Sequential	10	24297	0.89	86.3
Parallel	10	3411		
Sequential	100	242947	0.92	86.4
Parallel	100	33102		

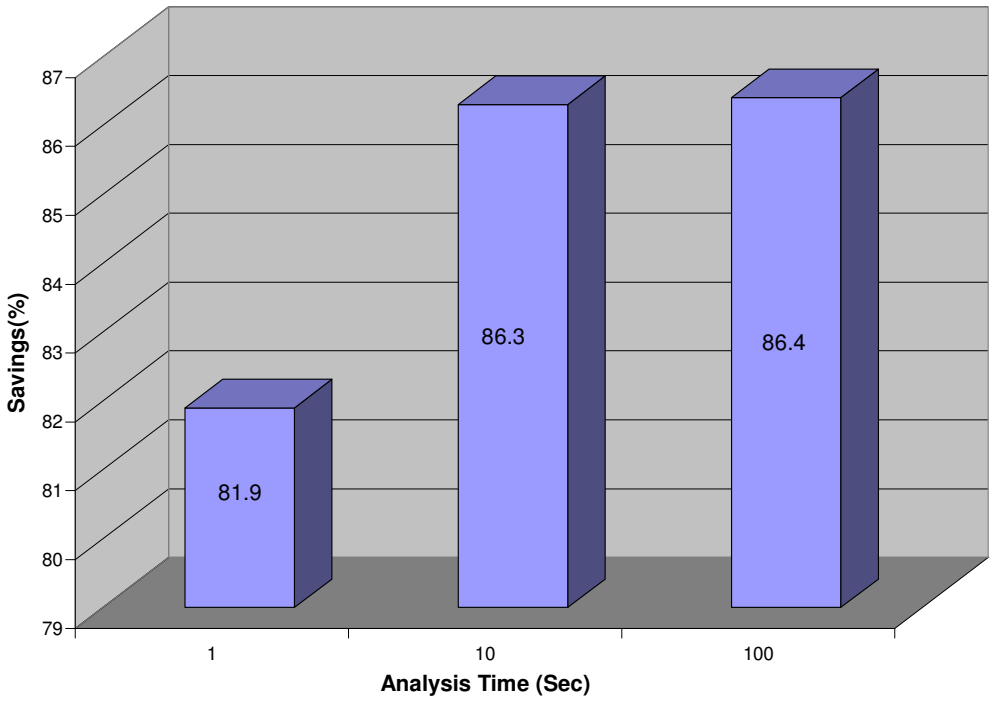

Figure 5.11: Savings (%) for N=8

Looking at the table 5.9 and Figure 5.12 it can be inferred that there is an increase in Efficiency with the increase in Analysis time.

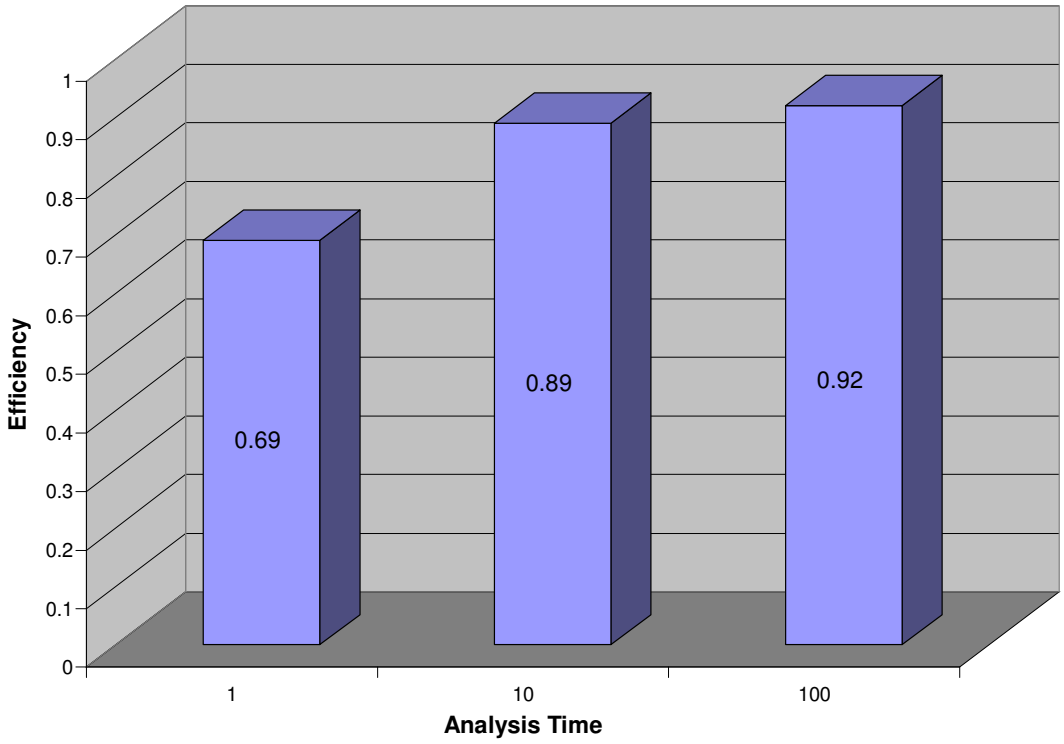

Figure 5.12: Efficiency for N=8

Results for a system with eight subsystems are similar to the system with four subsystems. Savings (%) and Efficiency are positively correlated with the Analysis Time. Savings (%) values are plotted in Figure 5.11 and Efficiency is plotted in Figure 5.12.

It is observed that while the values of Savings (%) are higher with eight subsystems, values of Efficiency are lower for each value of Analysis Time. A higher Savings (%) can be attributed to increase in parallel component of the code. Experimental results for the system with sixteen subsystems are similar to that of system with four and eight subsystems. Savings (%) and Efficiency are again positively correlated with the

Analysis Time. Savings (%) values are plotted in Figure 5.13 and Efficiency is plotted in Figure 5.14.

Table 5.10: Simulation results for N=16

Run Type	Time	Wall Time (Seconds)	Efficiency	Savings (%)
Sequential	1	4839	0.56	88.9
Parallel	1	539		
Sequential	10	48298	0.86	92.7
Parallel	10	3509		
Sequential	100	482883	0.91	93.1
Parallel	100	33202		

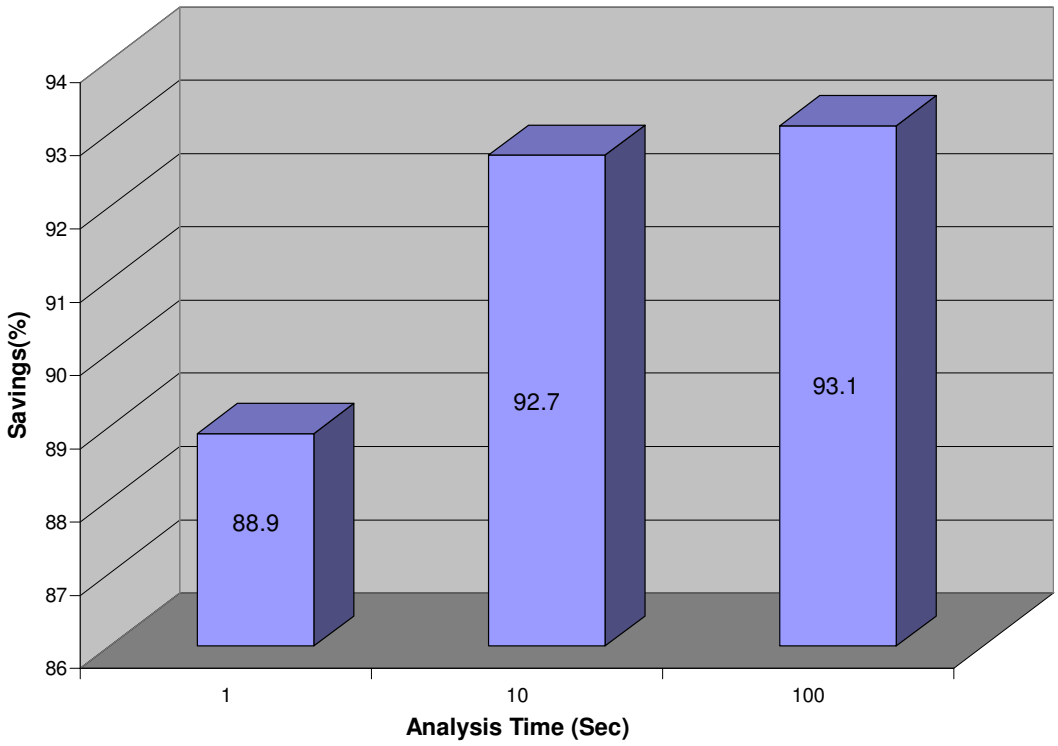

Figure 5.13: Savings (%) for N=16

Values of Savings (%) are again higher with sixteen subsystems than with eight subsystems. As discussed earlier the value of Savings (%) for eight subsystems was

higher than that of four subsystems. Values of Efficiency are lower for each value of Analysis Time. A higher Savings (%) can be attributed to increase in parallel component of the code.

The total Wall Time is again lower for the parallel execution as compared to the sequential execution. The difference between the sequential and parallel execution times is also increasing with more subsystems. This is the expected result because each subsystem is associated with a processor and more the number of subsystems larger the computing power is used.

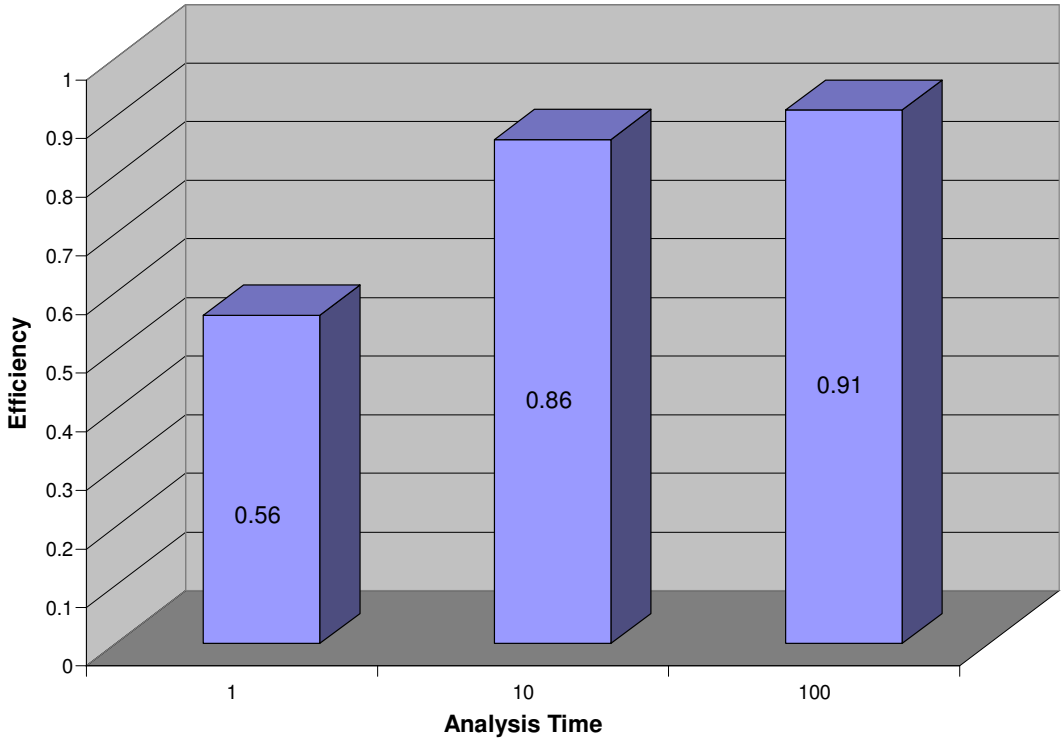

Figure 5.14: Efficiency for N=16

The values of Savings (%) for all the three cases of N = 4, 8, and 16 have been plotted in Figure 5.15. It is inferred that Savings (%) increases with the increase in number of

subsystems. An increase in Savings (%) with increase can be explained as the increase in parallel component of the code.

The analysis part is handled by one processor and remains constant for each experiment. The synthesis part varies with each experiment and larger the number of subsystem the larger will be the Analysis time. The total Wall Time is sum of Analysis time and synthesis time. For a given number of subsystems the Savings (%) increases with increase in complexity of the problem.

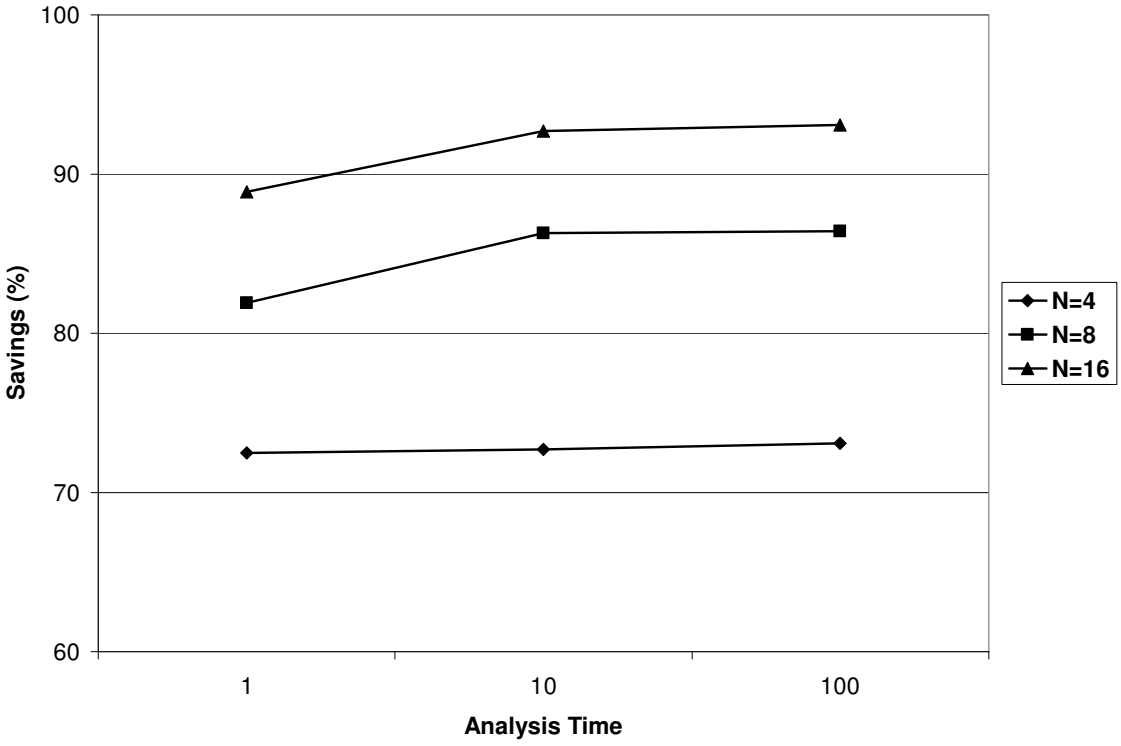

Figure 5.15: Comparison of Savings (%)

The values of Efficiency for the experiments with N=4, 8, and 16 are plotted in Figure 5.16. Efficiency decreases with increase in number of subsystem. This is opposite of the trend for Savings (%). For a given value of experiment Efficiency increases with

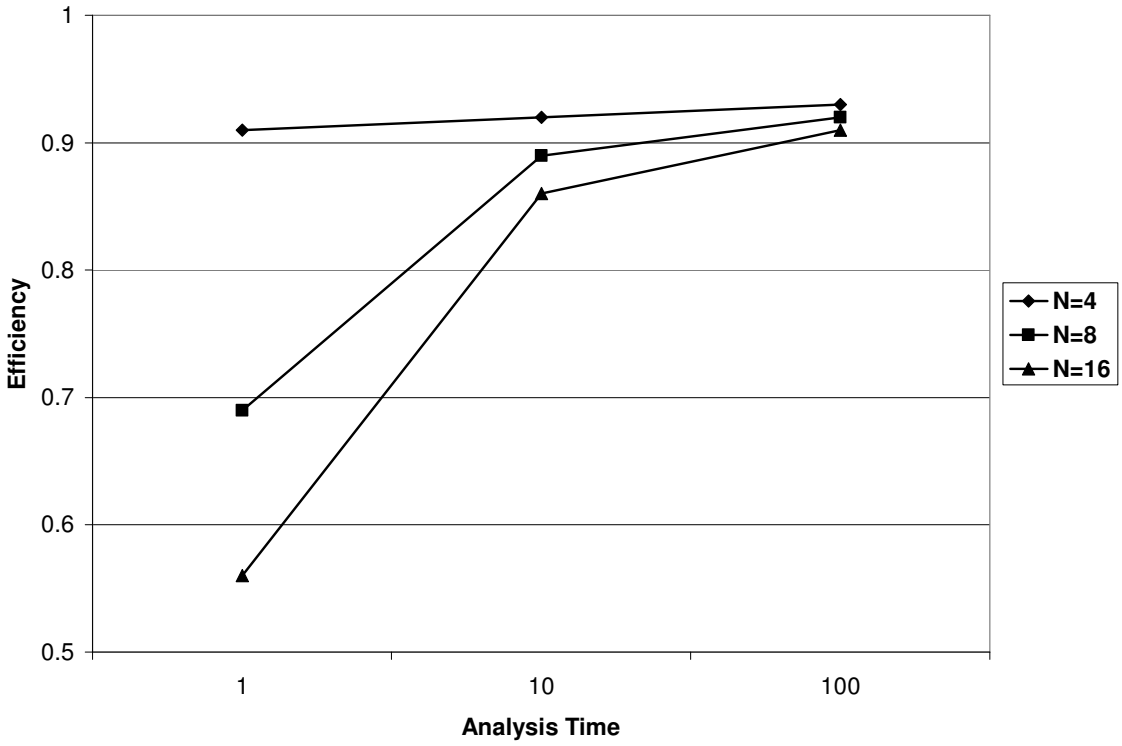

Figure 5.16: Comparison of Efficiency

increase in the complexity of the processor. This trend is the same as for the Savings (%). For each experiment the value of Efficiency seems to reach a steady state value. The steady state value is same, and it does not depend on the number of subsystems.

Theoretically the total Wall Time when Analysis Time = 1 second should be 330 sec without any communication but for parallel execution with exchange the value is found to be 339 seconds. Thus the time required for communication is 9 seconds. The same experiment was carried out with Time = 2 seconds and communication time was found to be 10 seconds.

For a given problem size communication time does not depend on the optimization time. The communication time depends on the number of subsystems and the amount of data transfer, but as the complexity of the simulation increases the communication becomes a very small fraction. As the complexity increases all systems tend towards the same value. Since the number of subsystems is not different by a large factor therefore all experiments gave the same final value of Efficiency.

5.2 Conclusions

From the results discussed in section 5.1 following conclusions can be drawn:

1. The increase in Savings (%) does not depend on the number of Analysis cycles or the length of experiment.
2. Wall Time deceases with parallel implementation for problems of all sizes (Number of subsystem). Larger the number of subsystems, higher will be the gain.
3. Savings (%) increases with increase in the number of sub systems. This is due to larger fraction of the work being simultaneously evaluated.

4. Savings (%) also increases with an increase in the optimization complexity of a problem. Optimization complexity is measured by the time required for running the subsystem level optimization once.
5. Savings (%) will be larger if the time required to optimize each subsystem is comparable. Although the decrease in Wall Time will be optimal if all the values are the same but, this will not be true in general for a real life problem

5.3 Future Work

MDO is a fast evolving research area with enormous opportunity for application in many fields. Present research has shown that DSDIDES can be successfully run in the parallel computing environment. This opens a new set of opportunities for future researchers. Some of the possible areas for future work are

1. In the present research DSIDES has been run on a parallel cluster without disturbing the DSIDES framework. There will be great opportunity to parallelize the DSIDES code. There are various stages where parallelization could be implemented. This will act like dual parallelization and hence can increase the speed by decreasing the time of simulation.
2. The proposed framework for implementing DSIDES in parallel can be tested for a large array of problems. The results of speed up attained can be compared with other similar approaches being adopted at other research universities to find the robustness of this framework.
3. Apart from DSIDES there are various optimizers available. Efforts could be directed towards running those optimizers in a parallel framework and comparing the results.

4. In the present research the subsystems where uncoupled, i.e. there were no shared variables. Shared variables pose a new challenge and will require a heuristic approach for sequencing or approximating. Research efforts can be directed towards parallelizing those problems and developing heuristics for the sequencing or variable approximation for each subsystem.
5. Implementing parallelization in a multi-level (more than 2 level) problem can be explored. In such a setting each subsystem will act as a system.
6. Combining different clusters for parallelization can provide enormous computing power. Such an effort can lead to a large reduction in run time if the data exchange between the clusters can be minimized.
7. Apart from Collaborative Optimization there are various other MDO frameworks, each with its unique set of properties. From the standpoint of parallel computation another framework that stands out is Concurrent Sub Space Optimization (CSSO). Research efforts can be directed towards implementing CSSO framework in a parallel computing environment.
8. Many organizations are working towards developing a web based application for running various MDO applications. Efforts can be directed towards creating such an application for the DSIDES environment.

References

[1] Amitay, I., Sudhakar K., and Mujumdar P. M., 2003, “Design and Development of MDO Framework”, MSO-DMES 2003 Conference Paper No. 78.

[2] Becker, J.C., Bloebaum, C.L., and Hulme, K.F., 1997, “Distributed computing for multidisciplinary design optimization using Java” Structural Optimization 14 (4): pp 203-218, DEC 1997.

[3] Braun, R. D., and Kroo, I. M., 1997, “Development and Application of the Collaborative Optimization Architecture in a Multidisciplinary Design Environment”, Multidisciplinary Design Optimization: State of the Art, N. Alexandrov and M. Y. Hussaini, Editors, SIAM.

[4] Chandy, K. Mani., and Stephen Taylor, 1992, “An Introduction to Parallel Programming”, Jones and Bartlett Publishers, Boston.

[5] Chen, W., Allen, J.K., Tsui, K-L, and Mistree, F., 1996, "A Procedure for Robust Design" ASME Journal of Mechanical Design, 118 (4), 478-485.

[6] Chen, W., Allen, J. K., Mavris, D., and Mistree, F., 1996, "A Concept Exploration Method for Determining Robust Top-Level Specifications" Engineering Optimization, 26, 137-158.

[7] Cramer, E.J., Dennis, J. E., Frank, P. D., Lewis, R. M. and Shubin, G. R., 1994, “Problem Formulation for Multidisciplinary Design Optimization”, SIAM Journal on Optimization, 4(4), pp. 754-776.

[8] Du, X and Chen, W., 2001, "A Most Probable Point Based Approach for Efficient Uncertainty Analysis", Journal of Design and Manufacturing Automation, 4(1), 47-66.

[9] Eldred, M. S., hart, W. E., Schimel, B. B., and van Bloemen Waander, B. G., 2000, “Multilevel Parallelism for Optimization on MP Computers: The Theory and Experiment”, In proceedings of the 8th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, AIAA-2000-4818, Long Beach, CA, Sept 6-8.

[10] Hamdi, M., Yi Pan, Hamidzadeh. B., F. M. Lim., 1997, “Potentials and limitations of parallel computing on a cluster of workstation”, International Conference on Parallel and Distributed Systems (ICPADS '97).

[11] Ignizio, J. P., 1985, "Multi-objective Mathematical Programming via the MULTIPLEX Model and Algorithm", European Journal of the Operational Research, 22, pp 338-346.

[12] Ignizo, J. P., 1985, "Introduction to Linear Goal Programming, Quantitative Applications in the Social Sciences", edited by J.J Sullivan and R.G. Niemi, Sage University Papers, Beverly Hills, CA.

[13] Internet document, <http://www.sgi.com/industries/manufacturing/mdo/index.html>, accessed on January 14, 2005.

[14] Kodiyalam, S. and Sobieszczanski-Sobieski, J., 2001, "Multidisciplinary design optimization – some formal methods, framework requirements, and application to vehicle design", Int. J. Vehicle Design (Special Issue), pp. 3–22.

[15] Kodiyalam, S., Sobieszczanski-Sobieski, J., 2001, "Alternate Sampling Methods for use with Multidisciplinary Design optimization", ASME.

[16] Kodiyalam, S., Yang, Lei Gu, R. J., and Tho, C.H., 2001, "Large-Scale, Multidisciplinary Optimization of a Vehicle System in a Scalable, High Performance Computing Environment", USACM.

[17] Manolache, F. B., and Costiner, S., 2002, "Parallel Processing Approaches for Multi Disciplinary Optimization Algorithms", CNA-report.

[18] McAllister, C. D., Simpson, T.W, 2003, "Multidisciplinary Robust Design Optimization of an Internal Combustion Engine", Transactions of ASME, Vol. 125.

[19] McAllister, C. D., 2002, "Uncertainty Propagation in Multidisciplinary Design Optimization", PhD Thesis.

[20] Mistree, F., Hughes, O.F., and Bras, B.A., "The Compromise Decision Support Problem and the Adaptive Linear Programming Algorithm", Ed. Kamat, M.P., Structural Optimization: Status and Promise, Washington, DC, (pp. 247-286), AIAA.

[21] Pacheco, P.S., 1997, "Parallel Programming with MPI", Morgan Kaufmann Publishers.

[22] Papalambros, P. Y., and Wagner, T. C., 1991, ''Optimal Engine Design Using Nonlinear Programming and the Engine Assessment Model,'' Ford Motor Company Scientific Research Laboratories & University of Michigan Department of Mechanical Engineering, 1991.

[23] Papalambros, P. Y., and Wilde, D. J., 2000, Principles of Optimal Design: Modeling and Computation, Cambridge University Press, New York.

[24] Renaud, J. E. and Gabriele, G. A., 1993, "Improved Coordination in Non-Hierarchic System Optimization," AIAA Journal, Vol. 31, Number 12, pp. 2367-2373.

[25] Renaud, J. E. and Gabriele, G. A., 1998, "Approximation in Non-Hierarchic System Optimization," AIAA Journal, Vol. 32, Number 1, pp. 198-205.

[26] Renaud, J. E., Apker B. T., and Perez M. V., 2002, "Parallel Processing in Sequential Approximate Optimization", AIAA-2002-1589

[27] [18] Simpson, T. W., 2003, "Multidisciplinary design optimization", Engineering and Technology Management, Aerospace America/December 2003, Page 38

[28] Simpson, T. W., Chen, W., Allen, J. K., and Mistree, F., 1997, "Designing Ranged Sets of Top-Level Design Specifications for a Family of Aircraft: An Application of Design Capability Indices," SAE World Aviation Congress and Exhibition, Anaheim, CA, AIAA-97-5513.

[29] Sobieszczanski-Sobieski, J., and Haftka, R. T., 1997, "Multidisciplinary Aerospace Design Optimization: Survey of Recent Developments," Structural Optimization, pp. 1-23, Vol. 14, No. 1.

[30] Sobieszczanski-Sobieski, J., Agte, J., and Sandusky, R., 1998, "Bi-Level Integrated System Synthesis (BLISS)," Proceedings, 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, AIAA, St. Louis, Missouri, September 1998. AIAA Paper No. 98-4916.

[31] Sobieski, I. P., and Kroo, I., 1998, "Collaborative Optimization using Response Surface Estimation", 36th AIAA Aerospace Sciences Meeting and Exhibit, Reno, NV, January 1998. AIAA Paper No AIAA-98-0915.

[32] Wujek B. A. , Renaud J. E. , Batill S.M., Brockman J. B. , 1996, "Concurrent subspace optimization using design variable sharing in a distributed computing environment", Concurrent Engineering – Research and Applications 4 (4): 361-377

Appendix A

Sysmain Subroutine

```
SUBROUTINE
sysmain(b,dI,dE,bgeo,dIgeo,dEgeo,dgeo,cr,w,bdyn,dIdyn,crdyn,wdyn,ddyn,sysflag)
!USE MPI
use vardec
IMPLICIT NONE
include 'mpif.h'
REAL,intent(in out) :: b,dI,dE,bgeo,dIgeo,dEgeo,dgeo,cr,w,bdyn,&
                dIdyn,crdyn,wdyn,ddyn
INTEGER :: status(MPI_STATUS_SIZE)
REAL, DIMENSION(0:7) :: msggeo
REAL,DIMENSION(0:9) :: msgdyn
Real :: sysflag

msggeo(7)= sysflag
msggeo(0) = b
msggeo(1) = dI
msggeo(2) = dE
msggeo(3) = bgeo
msggeo(4) = dIgeo
msggeo(5) = dEgeo
msggeo(6) = dgeo

CALL MPI_SEND(msggeo(0),8,MPI_REAL,2,tag,MPI_COMM_WORLD,ierr)
!print*, "i am sending to myid1 for subsys1"
!print*, b,dI,dE,bgeo,tag

msgdyn(9)  = sysflag
msgdyn(0)  = b
msgdyn(1)  = dI
msgdyn(2)  = cr
msgdyn(3)  = w
msgdyn(4)  = bdyn
msgdyn(5)  = dIdyn
msgdyn(6)  = crdyn
msgdyn(7)  = wdyn
msgdyn(8)  = ddyn


CALL MPI_SEND(msgdyn(0),10,MPI_REAL,3,tag,MPI_COMM_WORLD,ierr)

!print*, "i am sending to myid2 for subsys2"
```

```
!print*, b,dI,cr,w

CALL MPI_RECV(msggeo(0),7,MPI_REAL,2,tag,MPI_COMM_WORLD,status,ierr)

b     = msggeo(0)
dI    = msggeo(1)
dE    = msggeo(2)
bgeo  = msggeo(3)
dIgeo = msggeo(4)
dEgeo = msggeo(5)
dgeo  = msggeo(6)


CALL MPI_RECV(msgdyn(0),9,MPI_REAL,3,tag,MPI_COMM_WORLD,status,ierr)

b     = msgdyn(0)
dI    = msgdyn(1)
cr    = msgdyn(2)
w     = msgdyn(3)
bdyn  = msgdyn(4)
dIdyn = msgdyn(5)
crdyn = msgdyn(6)
wdyn  = msgdyn(7)
ddyn  = msgdyn(8)

!print*,"receiving value from myid 2 after completing the susbsystem 2 job"
!print*,b,dI,cr,w,bdyn,dIdyn,crdyn,wdyn,ddyn

END SUBROUTINE sysmain
```

Appendix B

Sequential Benchmarking Code

```
! Code developed by Shahab Nayyer
! Louisiana State University, December 2004
Module Sleeps
Implicit None
Integer,Public::Num_SubSystem,Num_Synthesis_Cycle,Num_Analysis_Cycle
Public::Optimizer
Contains
Subroutine Optimizer(Time_Optimize)
Integer :: Time_Optimize
    Call sleep(Time_Optimize)
End Subroutine
End Module sleeps

Program FACE
Use Sleeps
Use Iflport
Use Mpi
Implicit None
Integer::Synthesis_Time,Analysis_Time
Integer::I,J,K,Data_Input_File,Data_Output_File
Real(4):: Time_Elapsed
Real,dimension(1:1000,1:1000):: Array_Sys
Time_Elapsed = Timef()
Array_Sys = 999.0
Data_Input_File = 11
Data_Output_File = 12
Open(unit=Data_Input_File,File='data.inp',Status="Unknown",Action="Read")
Read(Unit=Data_Input_File,Fmt=*)
Read(Unit=Data_Input_File,Fmt=*)Num_SubSystem,Num_Analysis_Cycle,Num_Synth
esis_Cycle
Read(Unit=Data_Input_File,Fmt=*)
Read(Unit=Data_Input_File,Fmt=*)Synthesis_Time,Analysis_Time
Close(Data_Input_File)
Print*,Num_SubSystem,Num_Synthesis_Cycle,Num_Analysis_Cycle,Analysis_time,Sy
nthesis_time

Do I = 1,Num_Analysis_Cycle
        Call Optimizer(Analysis_time)
        Do J = 1,Num_SubSystem
                Do K = 1,Num_Synthesis_Cycle
                        Call Optimizer(Synthesis_Time)
                Enddo
```

```
        Enddo
Enddo
Open(unit=Data_Output_File,File='Soln.out',Status="Unknown",Action="Write")
Write(Unit=Data_Output_File,Fmt=*)'Num_SubSystem  Num_Analysis_Cycle
Num_Synthesis_Cycle'
Write(Unit=Data_Output_File,Fmt=*)Num_SubSystem,Num_Analysis_Cycle,Num_Syn
thesis_Cycle
Write(Unit=Data_Output_File,Fmt=*)'Synthesis_Time   Analysis_Time'
Write(Unit=Data_Output_File,Fmt=*)Synthesis_Time,Analysis_Time
Time_Elapsed = Timef()
Write(Unit=Data_Output_File,Fmt=*)'Total Wall Time(Seconds)= ',Time_Elapsed
Close(Data_Output_File)

!Print*,Num_SubSystem,Num_Synthesis_Cycle,Num_Analysis_Cycle,Analysis_time,Sy
nthesis_time
!Print*,'Total Wall Time(Seconds) =',Time_Elapsed
End program
```

Appendix C

Parallel Benchmarking Code

```
! Code developed by Shahab Nayyer
! Louisiana State University, December 2004
Module PSleeps
Use MPI
Implicit None
Public::Sub_Optimizer,Sys_Optimizer
Integer,Public ::
MyId,NumProcs,Num_SubSystem,Num_Analysis_Cycle,Num_Synthesis_Cycle
Integer,Public::Synthesis_Time,Analysis_Time,X_Dim,Y_Dim

Contains

subroutine Sys_Optimizer(Proc_Id,Time_Optimize)
Integer :: Proc_Id,Time_Optimize,Ierr,Tag,I,J,K
INTEGER :: status(MPI_STATUS_SIZE)
Real(8),Dimension(1:X_Dim,1:Y_Dim):: Array_Sys
Array_Sys = 999.0
Tag = 99
print*,MyId,'Sys'

        Do I = 1,Num_Analysis_Cycle
                Call Sleep(Time_Optimize)
                Do J=1,Num_SubSystem
                Call
Mpi_Send(Array_Sys(1,1),X_Dim*Y_Dim,Mpi_Real,J,tag,Mpi_Comm_World,ierr)
                !Print*,'send data to',j
                Enddo
                Do K=1,Num_SubSystem
                Call
Mpi_Recv(Array_Sys(1,1),X_Dim*Y_Dim,Mpi_Real,K,tag,Mpi_Comm_World,status,ie
rr)
                Enddo
        Enddo

End Subroutine Sys_Optimizer

Subroutine Sub_Optimizer(Proc_Id,Time_Optimize)
Integer :: Status(Mpi_Status_Size)
Integer :: Proc_Id,Time_Optimize,I,J,Ierr,Tag
Real(8),Dimension(1:X_Dim,1:Y_Dim):: Array_Sub
Tag = 99
Print*,Myid,'sub'
```

```
        Do I=1,Num_Analysis_Cycle
        Call
MPI_Recv(Array_Sub(1,1),X_Dim*Y_Dim,Mpi_Real,0,Tag,Mpi_Comm_World,Status,I
err)
                Do J=1,Num_Synthesis_Cycle
                        Call sleep(time_optimize)
                Enddo
      !Print*,myid,'Reveived data'
        Call
Mpi_Send(Array_Sub(1,1),X_Dim*Y_Dim,Mpi_Real,0,Tag,Mpi_Comm_World,Ierr)
      Enddo
End Subroutine Sub_Optimizer


End Module PSleeps

Program Face
Use PSleeps
Use Mpi
Use Iflport
Implicit None

Integer::Ierr,Counter,I,Data_Input_File,Data_Output_File
Real(4) ::Time_Elapsed
Counter = 0
Data_Input_File = 21
Data_Output_File = 22
Time_Elapsed = Timef()
Call Mpi_Init(Ierr )
CALL Mpi_Comm_Rank( Mpi_Comm_World,MyId,Ierr )
CALL Mpi_Comm_Size( MPI_Comm_World,NumProcs,Ierr)
If (Myid == 0) then
        Open(unit=Data_Input_File,File='data.inp',Status="Unknown",Action="Read")
        Read(Unit=Data_Input_File,Fmt=*)
        Read(Unit=Data_Input_File,Fmt=*)Num_SubSystem,Num_Analysis_Cycle,Num
_Synthesis_Cycle
        Read(Unit=Data_Input_File,Fmt=*)
        Read(Unit=Data_Input_File,Fmt=*)Synthesis_Time,Analysis_Time
        Read(Unit=Data_Input_File,Fmt=*)
        Read(Unit=Data_Input_File,Fmt=*)X_Dim,Y_Dim
        Close(Data_Input_File)
EndIf
Call Mpi_Bcast(Num_Analysis_Cycle,1,Mpi_Integer,0,Mpi_Comm_World,Ierr)
Call Mpi_Bcast(Num_Synthesis_Cycle,1,Mpi_Integer,0,Mpi_Comm_World,Ierr)
Call Mpi_Bcast(Synthesis_Time,1,Mpi_Integer,0,Mpi_Comm_World,Ierr)
Call Mpi_Bcast(X_Dim,1,Mpi_Integer,0,Mpi_Comm_World,Ierr)
```

```
Call Mpi_Bcast(Y_Dim,1,Mpi_Integer,0,Mpi_Comm_World,Ierr)
Print*, 'Before Wait'

If (MyId == 0) then
        Call Sys_Optimizer(MyId,Analysis_time)
Else
        Call Sub_Optimizer(MyId,Synthesis_Time)
EndIf

!Print*,'After Wait'
Time_Elapsed = Timef()
Call Mpi_Barrier(Mpi_Comm_World,Ierr)

If (Myid== 0) then
        Open(unit=Data_Output_File,File='PSoln.out',Status="Unknown",Action="Write
")
        Write(Unit=Data_Output_File,Fmt=*)'Num_SubSystem  Num_Analysis_Cycle
Num_Synthesis_Cycle'
        Write(Unit=Data_Output_File,Fmt=*)Num_SubSystem,Num_Analysis_Cycle,Nu
m_Synthesis_Cycle
        Write(Unit=Data_Output_File,Fmt=*)'Synthesis_Time   Analysis_Time'
        Write(Unit=Data_Output_File,Fmt=*)Synthesis_Time,Analysis_Time
        Time_Elapsed = Timef()
        Write(Unit=Data_Output_File,Fmt=*)'Total Wall Time(Seconds)=
',Time_Elapsed
        Close(Data_Output_File)
EndIf

Call Mpi_Finalize(Ierr)
End Program
```

Vita

Shahab Nayyer received his Bachelor of Technology degree in chemical engineering from Indian Institute of Technology (IIT)-Chennai, India, in May 2002. He joined the department of Industrial Engineering at Louisiana State University (LSU), Baton Rouge, Louisiana, in spring 2003 as a graduate student. Before joining LSU, he worked as a Program Analyst at Cognizant Technology Solutions-Chennai, India. He has worked in the areas of Simulation, Information Systems, Reliability Engineering, Supply Chain Management and Optimization during his graduate program in Industrial Engineering at LSU. He has extensively worked in parallel computing in the areas of modeling and optimization. He is also enrolled in the Master of Science program in the Department of Finance at LSU. He is a candidate for the degree of Master of Science in Industrial Engineering and Master of Science in Finance to be awarded at the commencement of May 2005.