DEVELOPMENT OF *CASCADE*: A MULTIDISCIPLINARY DESIGN TEST SIMULATOR

K. F. Hulme* and C. L. Bloebaum†
Multidisciplinary Optimization and Design Engineering Laboratory (MODEL)
Department of Mechanical and Aerospace Engineering
State University of New York at Buffalo

Abstract

Industry is constantly pursuing faster and cheaper means for designing and manufacturing high quality goods and services. This becomes ever more difficult as the complexity of the product increases. Much of the method development in the field of Multidisciplinary Design Optimization (MDO) attempts to simplify the design of a large, complex system, by dividing the system into a series of smaller, simpler, and coupled subsystems. These methodologies require extensive testing, either on analytical representations of real-life systems, or on the actual system itself, prior to implementation in order to verify their feasibility and robustness. Due to the complexity of these real-life systems, however, it is not practical to perform methodology feasibility studies on the analytical and numerical representations of the true system. Hence, some representative yet efficient means of determining the feasibility and robustness of MDO methods is crucial. This paper describes the design of a test simulator, CASCADE (Complex Application Simulator for the Creation of Analytical Design Equations), that is capable of randomly generating and then converging a complex system of analytical equations, of user-specified size. CASCADE-generated systems can be used to test sequencing and system reduction strategies, convergence strategies, and MDO methods, among others. Further, CASCADE has been designed to operate in a distributed computing environment (using Parallel Virtual Machine), as the field of MDO itself is perfectly suited for such a setting. This paper describes the simulator and many of its applications.

Background

The design methodologies used in worldwide industry have been changing rapidly. United States aircraft, automotive, and electronics industries, among others, are constantly searching for ways to improve the efficiency of their design processes to meet time and cost demands. The traditional serial design approach is characterized by a sequential design cycle, where a design is formulated in a given design group and passed to the next group, who uses the previous groups' output parameters as input parameters. Because of both inefficiency and design system complexity, this approach has largely become obsolete, in favor a concurrent design approach.

It is desired to achieve a systematic approach to the integrated, concurrent design of products and their related processes, including manufacture and support. The interaction of all participating engineering groups throughout the design cycle is a truly *multidisciplinary* effort. Many of the recently developed capabilities to address concurrent design have stemmed from the field of Multidisciplinary Design Optimization (MDO). The MDO approach is intuitive in that one often attempts to divide one large task into a group of smaller, interrelated (coupled), and more manageable tasks [17]. The large task is often referred to as a *system*, and the smaller, interrelated tasks are called *subsystems*. Each subsystem contains *design variables*, as well as additional unknown outputs, often referred to as *behavior variables*. These subsystem variables are collectively referred to as *modules* of the system. Each subsystem can be thought of as a participating design group of a large scale design. An example would be the *aerodynamics* division of the design of an aircraft. One goal of MDO is to analyze these subsystems concurrently, thus speeding up the design time of the overall product. This method was first established by applying a linear decomposition method to a hierarchical (top-down) system [19,1].

Most design cycles contain participating groups that interact laterally. Such design cycles are thus non-hierarchic in nature. The Global Sensitivity Equation (GSE) method was the first approach to extend the concepts of the linear decomposition method to non-hierarchical systems [19,1]. This method uses *local* sensitivities (derivatives that are computed within each subtask) to compute total system sensitivities.

An MDO issue that is the focus of much research today is the concept of scheduling (or

* Research Assistant.
† Associate Professor, Member AIAA.

sequencing) the coupled subsystems. Scheduling is a methodology that reorders the design tasks (modules) in a given system, to allow for maximum efficiency in the execution of the design [15]. The efficiency of a system can be increased if certain problem-dependent parameters are minimized, such as cost, CPU time, feedbacks, or crossovers. A feedback occurs when a system module requires information from another module that is located later in the design sequence. A crossover occurs when the feedbacks of two modules intersect, without any transfer of information.

Another area of current research within the field of MDO is the concept of system reduction, through coupling suspension and elimination [3,17]. The decomposition of a large system results in a series of smaller subsystems, that are interrelated through couplings. Because of the enormity of many engineering systems, there is a need to minimize the complexities of the system, and thus the time for both design system convergence and sensitivity analysis. It is, hence, advantageous to find an analytical means for quantifying the strengths of these couplings. Couplings that are found to be weak can be suspended for a portion of the system analysis, or eliminated outright. This concept provides the foundation for system reduction strategies.

Industry is primarily interested in the way that the various participating design groups communicate. An efficient and natural way for design groups to pass information back and forth is via distributed processing. The concept of distributed processing assures that the system design tasks are computationally distributed among the participating design groups. In this way, distributed processing extends the principals of MDO to a computer network. This methodology allows for parallel communication between the design groups, and hence provides greater efficiency than a sequential computing approach. A modern day approach that is used to achieve distributed processing is the Parallel Virtual Machine (PVM) coding language [9]. PVM uses library calls and message passing to distribute tasks amongst the individual computer hosts on the network.

Motivation for Creation of CASCADE

It is quite clear that the field of MDO has a great deal of potential to provide methodologies that can be used in industry. For this to happen, these MDO-methodologies must be tested extensively. Researchers must typically spend a great deal of time and effort to develop complex systems to test their methodologies. These systems are often quite large and computationally expensive to deal with. It would, therefore, be convenient for these researchers to possess a simulator that is capable of generating, converging, and then further analyzing an analytical representation of a complex engineering system. The simulator should be robust and capable of representing a wide range of complex systems. The simulation should likewise be arbitrary (random), as many design scenarios are often presented with issues that were initially unpredicted. Lastly, the simulation should be realistic, in that it should allow for a distributed processing communication architecture.

Given the above motivation, this paper discusses the design and creation of an MDO-type simulator, *CASCADE* (Complex Application Simulator for the Creation of Analytical Design Equations). CASCADE can be used to generate complex systems comprised of analytical equations, of user specified size. Thereafter, CASCADE employs a system analysis to iteratively converge the generated system. To add realism to the simulation, this process can be made to take place in a distributed environment, using the PVM coding language. After the system has converged, CASCADE uses the GSE method to compute the total sensitivities off all output responses, with respect to all inputs (design variables). This sensitivity information could potentially be used to analyze coupling strengths for possible suspension/elimination or could be used in an optimization implementation. CASCADE writes each converged output (behavior variable) equation to a separate subroutine. Researchers could potentially experiment with this sequence of subroutines to further investigate coupling strengths, sequencing issues, convergence strategies, or optimization methods.

Programming Methodology of CASCADE

The CASCADE simulator is described by first addressing the make-up of the coupled system and the inputs required by the user, then discusses the means of achieving a converged system, and finally addresses post-convergence features.

System Construction

CASCADE has been designed to create unpredictable, randomly generated systems of user-prescribed size that will best represent the wide variety of MDO problems that might be of interest to researchers. For this reason, a random number generator is used to make a number of system-related decisions, such as the number of terms per behavior variable (output equation), the value of the coefficient associated with each term, and the number of design variables per subsystem. Prior to executing CASCADE to generate and converge a system, the user must

execute an input file to tell CASCADE the size of the system to be generated and to specify a variety of other options, pertaining both to file/screen writing and convergence. Once these preliminaries are taken care of, the system can be constructed, term by term.

The terms that are generated and combined to create the coupled set of equations take the form:

$$y_i = \sum a_j x_j^{b_j} + \sum c_k w_k^{d_k} + \ldots \qquad [1]$$

where, y_i is the ith output of the Y subsystem, and is a function of i design variables x and output couplings from other subsystems such as w_k's from subsystem W. Further, each of these design variables and coupling outputs can be raised to some power (b_j and d_k) and is premultiplied by a coefficient (a_j and c_k).

The *determ* subroutine determines the nature of every term in the system, on the first iteration of execution. For each term in the system, the sign, exponent (one of a possible six choices), and coupling nature (coupling to either a design variable or to a behavior variable) of each term is determined, in this subroutine. Next, the magnitudes of each term and the sensitivity of each term are determined in the *mags* subroutine. Design variable magnitudes are identically known as they are determined initially by the random number generator. Hence, the magnitudes of design variable-coupled terms can be identically computed on the first iteration. CASCADE generates non-hierarchical systems which require both initial guesswork of the behavior variable magnitudes and iteration to converge. For this reason, the magnitudes of behavior variable-coupled terms cannot initially be computed identically, as the magnitudes of the behavior variables are not initially known. The initial guess for the magnitude of all behavior variables, on the first iteration, is chosen to be 1.0. The magnitude of every term in the system is limited, so as to prevent a diverging behavior variable. This limitation process takes place in the *termmag* subroutine. Next, equation magnitudes are computed by simply adding the magnitudes of every term, that comprises each equation of the system. This magnitude (for all behavior variables) is indirectly limited by the *termmag* subroutine, and as a check, is directly limited by the *eqnrange* subroutine. An upper bound of 9999.0 has been set for every behavior variable, so as to prevent a diverging equation. A lower limit of 0.0 is set for every behavior variable, so as to prevent undefined exponentiations of fractional powers. (Three of the six exponent choices in the *determ* subroutine are fractions). These various settings were determined heuristically based on a number of stochastic runs of various systems.

System Convergence

Once all equation magnitudes have been determined and found to be within an "acceptable" range, a convergence check takes place, in the subroutine *converge*. The newly computed magnitudes of each behavior variable, on the present iteration, are compared to the corresponding magnitudes from the previous iteration. If the differences between these values, for all behavior variables, are less than or equal to the initial convergence criteria, then the system is determined to be converged. If not, the process repeats, and the newly computed (and more accurate) behavior variable magnitudes are used for the next iteration. "Convergence ease" parameters have been incorporated into the program, such that the user can ease the convergence criteria by 1 and/or 2 decimal places after a user-defined number of iterations. This will take place if the system is having problems converging to the initial convergence criteria.

Post-convergence features

Once the system has converged, CASCADE performs a variety of post-convergence features, that could be greatly beneficial to an MDO-researcher. The *eqnsubs* subroutine uses character strings to write every converged behavior variable of the created system to a separate subroutine. This process can take place in either the FORTRAN or the ANSI C computer languages. These subroutines could be used in a system reduction analysis, to view the effects of the suspension and/or elimination of relatively weak couplings. Alternatively, each converged subroutine could be optimally sequenced, and then re-converged, to test sequencing or convergence strategies. Also, these subroutines can be used to test distributed computing approaches.

The *derivs* subroutine uses the GSE approach to attain the total derivative matrix of the system that has been created. The left and right hand side partial derivative matrices seen in equation [2] can be computed, as the sensitivity of each output equation of the system can be computed, with respect to both a) every other behavior variable and b) every subsystem design variable.

$$\begin{bmatrix} 1 & -\frac{\partial Y_A}{\partial Y_B} \\ -\frac{\partial Y_B}{\partial Y_A} & 1 \end{bmatrix} \begin{bmatrix} \frac{dY_A}{dX_A} & \frac{dY_A}{dX_B} \\ \frac{dY_B}{dX_A} & \frac{dY_B}{dX_B} \end{bmatrix} = \begin{bmatrix} \frac{\partial Y_A}{\partial X_A} & 0 \\ 0 & \frac{\partial Y_B}{\partial X_B} \end{bmatrix} \qquad [2]$$

These matrices are then normalized, and an LU-Decomposition is used to attain the normalized total derivative matrix. The total derivatives are recovered by "un-normalizing" each element of this matrix. This matrix can be used to assess the coupling relations that are weak relative to the others. It is possible that these weak couplings be eliminated from the system analysis, or at least suspended for part of it. These derivatives could also be used for some form of formal system optimization procedure. Finally, the *param* subroutine provides a statistical output listing of the constructed system. This subroutine provides the cpu times required to both a) build and converge the system and b) compute the total derivative matrix, and lists the number of iterations required for convergence of the created system. This file also provides a term by term listing of the nature of each term in the system, as determined earlier by the *determ* subroutine. This latter feature could be easily modified to allow for the perturbation and *re-entry* of a previously generated system into any of a variety of system improvement software packages. This then assures that systems can be reproduced for method comparison studies.

Extension to PVM

The above procedure was initially designed for single computer operation, and was later modified to operate in a distributed computing environment, via PVM. A *master* machine enrolls in PVM, and performs the preliminaries: namely, random number generation, and reading in the input file. The master then spawns *slave* tasks on the other *hosts* that were detected on the *virtual machine* that is presently available. The master then packs and sends array data for one subsystem, to each slave. Each slave receives and unpacks the data sent to it by the master, and then performs the *determ*, *mags*, *termrange*, and *eqnrange* subroutines, for the one subsystem that was sent to it. Each slave then sends the newly computed data back to the master for a convergence check. The process repeats until the system converges. Post-convergence features take place on the master machine, and the slaves exit their processes.

Sample CASCADE system

In this sample system, the user has decided that the system will have 3 subsystems. The user has also decided that subsystem 1 has two outputs "w_1"and "w_2", subsystem 2 has four outputs "y_1", "y_2", "y_3", and "y_4", and subsystem 3 has three outputs "z_1", "z_2", and "z_3" (Figure 1).

Recall that CASCADE will randomly determine the number of terms per output equation (ranging from 1 to 20), the number of design variables per subsystem (ranging from 1 to 5), as well as the coupling nature, coefficient, sign, and exponent of each term in each equation. Assume that CASCADE has decided that subsystem 1 will have 4 design variables, x_{11}, x_{12}, x_{13}, and x_{14}, and that equation w_1 (of subsystem 1) will have 3 terms. Table 1 summarizes CASCADE's decisions that lead to the construction of equation w_1.

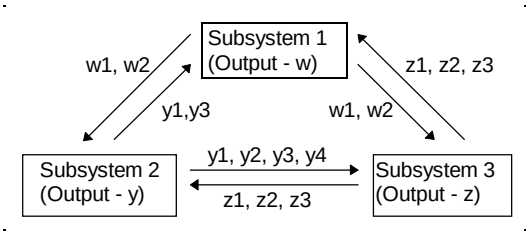

Figure 1. System of 3 coupled subsystems.

Table 1. CASCADE term generation for equation w_1

equation w_1	term 1	term 2	term 3
sign	positive	positive	negative
coefficient	0.967	0.265	0.087
coupling nature	design variable(x)	behavior variable(y)	behavior variable(z)
coupling number	3	4	1
exponent	2	-1	1/3

As a result, equation w_1 will appear as follows:

$$w_1 = + 0.967x_{13}^{2} + 0.265y_4^{-1} - 0.087z_1^{1/3} \qquad [3]$$

CASCADE carries out a similar procedure for equation w_2 of the same subsystem, as well as equations "y" and "z" of subsystems 2 and 3, respectively. Upon creation of the system, iterative convergence procedures commence, as previously described.

Outputs of CASCADE for varying system sizes

Since the thrust of this work focuses on the development of the CASCADE simulator, it is difficult to discuss results in the traditional sense. The worth of this simulator can only be measured in its use by the MDO community. It should be noted, however, that CASCADE has already been used extensively for optimal sequencing and convergence strategy studies, as well as for system reduction and for MDO method

testing. In this paper, however, the potentials of distributed computing using PVM are explored and discussed in some detail, with resulting computational times presented.

Clock time normalized per iteration

The clock time required to build and converge a complex system is a primary concern of a system analyst. CASCADE has been used here to generate a wide variety of systems and to analyze the execution times involved. This was done in an effort to better understand the potential advantages, as well as the drawbacks of distributed computing. All execution times have been normalized by the number of iterations that were required to converge the system to the specified criterion of 1.0E-4. The first result is seen in Figure 2, which plots the "single computer" results. The computers that were used are called Moriarty and EAS00; the former is a 12 processor mini-supercomputer, and the latter is a Sun-Sparc 4 workstation.

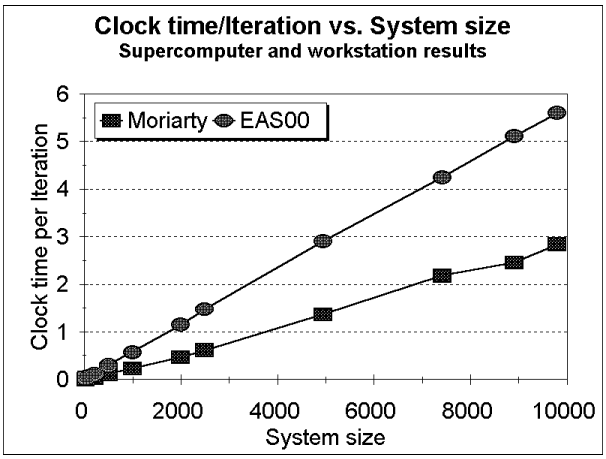

Figure 2. Single computer clock time results.

As expected, the supercomputer substantially outperforms the workstation. The supercomputer normalized clock times are at least twice as fast, for most system sizes. Most systems require from 40-60 iterations to converge, so the absolute clock times for the supercomputer are on the order of 180 seconds, for a 10000 equation system. Moreover, there is a near linear correlation between normalized clock time and system size, for both computer scenarios.

From several points of view, the distributed computing data of Figure 3 does not live up to expectations. Here, 3 scenarios are investigated, with 5, 10, and 24 slave machines. Both the master and slave machines were Sun Sparc 4 workstations, similar in nature to EAS00. As expected, there is a general upward trend in normalized clock time, with an increase in system size, for all 3 scenarios (5, 10, and 24 slave machines). However, the clock times are *larger* than those corresponding to the single machine data of Figure 2. At first glance, this appears to be counter-intuitive. One would think that a system that is solved by multiple machines would be constructed and converged more quickly than a system that is analyzed on one machine. This was not found to be the case.

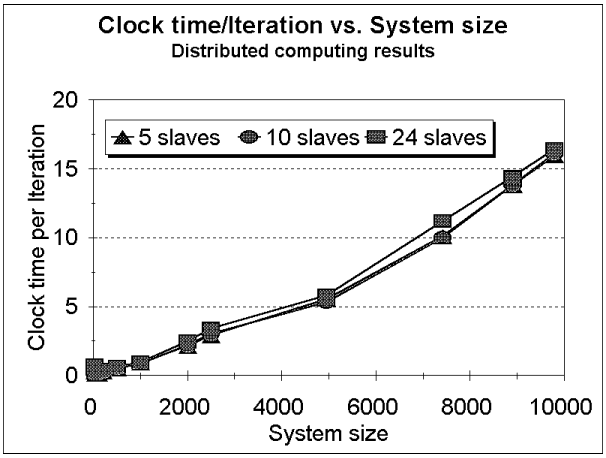

Figure 3. Distributed computing clock time results.

Upon comparing the three curves themselves in Figure 3, one sees that for the most part, systems of all sizes required roughly the same amount of time to converge, regardless of the number of slave machines being used. This again appears to be counter-intuitive. It would seem likely that a system solved using distributed computing techniques would converge faster, when using a greater amount of slave computers on the virtual machine. This was not found to be the case. Clearly, the question then becomes - - "Why?".

As "complex" as the system of equations that CASCADE generates are, the systems are not complicated enough to fully exploit the strengths of distributed computing. *Message passing* dominates the clock time involved with converging a system when using distributed computing techniques. To explore this situation further, "sleep" times were introduced into each subsystem analysis. In other words, each subsystem analysis is performed (either in a sequential or a distributed computing environment), and then the program execution pauses for a user-specified number of seconds. This sleep time is used to simulate a computation with a higher level of complexity.

Effects of sleep time on each subsystem iteration

Sleep times were initially experimented with on the Moriarty supercomputer, the EAS00 workstation, and the 5 and 10 slave distributed computing scenarios. Figure 4 is a plot of clock time vs. system size for all four scenarios, with a 0.01 second sleep time.

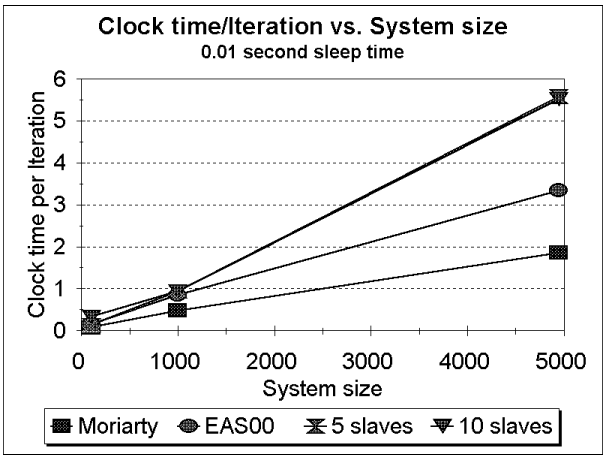

Figure 4. Normalized clock time - 0.01 sec sleep time.

This plot varies little from the results previously presented. Moriarty is still the fastest option for all system sizes, followed by EAS00, followed by both the 5 and 10 slave distributed computing scenarios, which vary little from each other.

Figure 5 is a similar plot, but the sleep time has been increased to 0.05 seconds. From this plot, important results can be seen. For smaller system sizes, the distributed computing scenarios outperform both Moriarty and EAS00. However, the differentiation between the 5 and 10 slave scenarios is still minimal. For large system sizes, Moriarty and (barely) EAS00 overtake the distributed computing scenarios and outperform them.

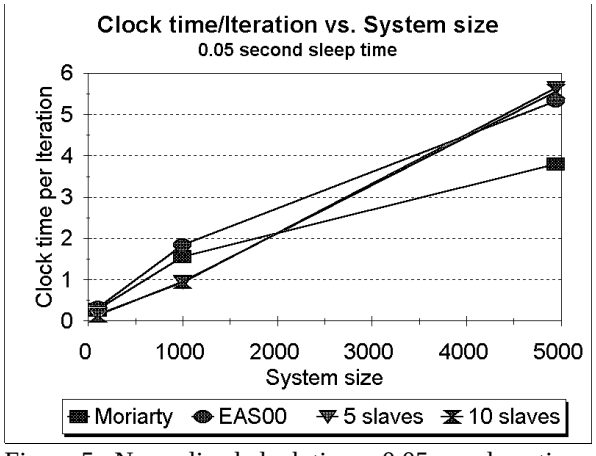

Figure 5. Normalized clock time - 0.05 sec sleep time.

Figure 6 is yet another plot of the same nature, this time with a 0.1 sleep time. With this amount of sleep time, the distributed computing scenarios clearly overtake both Moriarty and EAS00, for the full range of system sizes. However, there is still only minimal differentiation between the 5 and 10 slave scenarios.

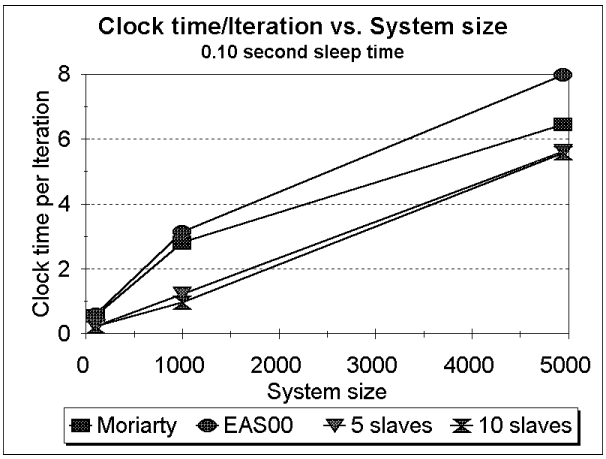

Figure 6. Normalized clock time - 0.10 sec sleep time.

To see if *any* numerical sleep time would have an impact on the variation between distributed computing results with varying numbers of slave machines, the sleep time was further increased to 0.5 seconds. The results are seen in Figure 7.

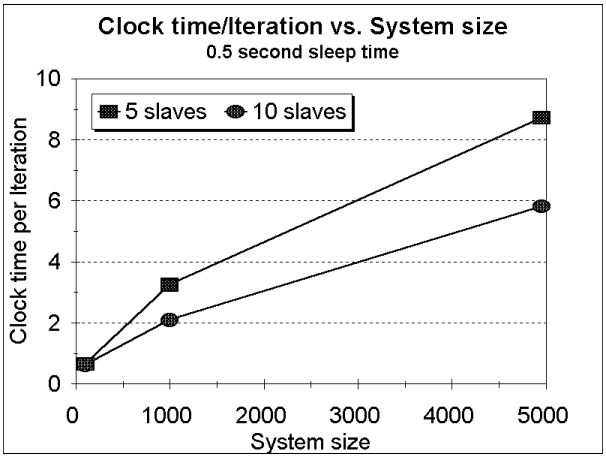

Figure 7. Normalized clock time - 0.5 sec sleep time.

Finally, the 5 and 10 slave machine curves split apart, with the 10 slave scenario clearly outperforming the 5 slave scenario. This trend is further accentuated in Figure 8, which plots similar results with a full 1 second of sleep time.

Hence, the subsystem analyses of the CASCADE-generated systems were forced to require a half to one second longer so that the benefits of

distributed computing could be fully demonstrated. This is an extremely important point for those who wish to use CASCADE to simulate a distributed computing environment. It is necessary to have sufficiently large sleep times so that message passing does not dominate and skew results. Of additional interest is the calculation of system derivatives, which can be used in a variety of MDO applications.

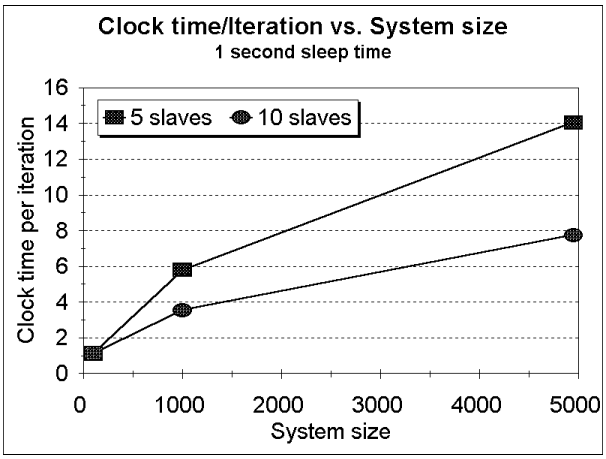

Figure 8. Normalized clock time - 1.00 sec sleep time.

Effects of Normalization on GSE solution

The sensitivity analysis of the converged system can be very useful to system-reduction researchers. The matrix of total derivatives provides an indication as to how sensitive each system output is to each system input. Normalization techniques are often used, so as to improve the numerical condition of the left and right hand side matrices that are used in the LU-decomposition, to attain the total derivative matrix. The benefits of this procedure can be seen from Figure 9.

Figure 9 is a plot of the clock time required to compute the total sensitivity matrix vs. the size of the system. For smaller systems, the clock times of the normalized and not-normalized systems are comparable. As the system size increases, the clock time required to compute the sensitivity matrix is slightly shorter for the normalized matrices than for the not-normalized matrices. This reduction in clock time is likely attributable to the numerical conditioning provided by the normalization procedure.

Table 2 lists *condition number* vs. system size, for both normalized and not-normalized scenarios. (The condition number provides a general indication as to how well-conditioned the partial derivative matrices were, when used to attain the total derivative matrix. Thus, the condition number reflects the reliability of the numerical estimate that was attained, for the total derivative matrix. A *low* condition number is desirable). The differences in magnitude are astounding. The normalized data has matrix condition numbers that are far lower than those for the not-normalized data. Using not-normalized data, a total derivative solution could not be found for the 990 equation system example, as depicted in the table. For this system size, the condition number is *infinite*, for all intents.

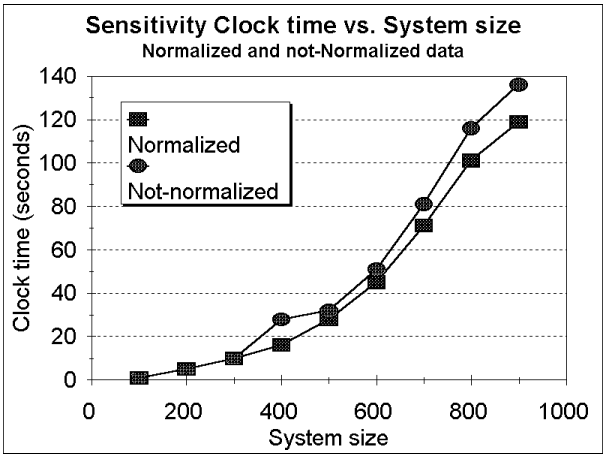

Figure 9. Sensitivity CPU times vs. system size.

There does not seem to be any correlation between condition number and system size. A large condition number will result when coupling complexity is *high*. The coupling nature of these systems is random, hence systems of high complexity (and hence, high condition number) can arise for any system size.

Table 2. Condition number comparison of normalized / non-normalized data

number of system equations	condition number normalized sensitivities	condition number Non-normalized sensitivities
100	40.06	60634.
200	21.41	37663.
300	37.10	101310.
400	81.06	46630.
500	390.70	101182
600	24.99	107055
700	60.09	58305.
800	53.48	259539.
900	60.32	68528.
990	345.83	47565084.**

** : no solution attained

Summary and Discussion

The field of Multidisciplinary Design Optimization inherently has great potential for becoming an industrial standard for the design of large, complex systems. The basic methodology is simple: divide a large task or system into numerous smaller and inter-related subtasks. These subtasks can be divided among the participating design groups and solved simultaneously, in a non-hierarchic manner. The analysis of a complex system must take place on a powerful computer. Hence, it is clear that the MDO methodology lends itself well to distributed computing environments, in which one large computational task is divided among numerous computers, connected on a network.

The motivation for MDO is to reduce the time and cost required for the design process. Most complex systems are non-hierarchic in nature, and require an iterative scheme (and initial approximations) to converge. Task sequencing researchers attempt to find the optimal sequence (order) in which to analyze the system modules, to gain the converged solution in the least amount of time. System reduction researchers seek to effectively reduce the size of a complex system, with a minimal loss in accuracy. Researchers have attempted to temporarily suspend and/or permanently eliminate output couplings that were comparatively found to be less substantial.

Before these MDO-strategies can be implemented in the design process of such large systems as automobiles and aircraft, they must be tested. A method for analytically simulating real-life, large system couplings is necessary. The simulation should be able to predict the output sensitivities of the system; the change in the system outputs with respect to a prescribed change in the design variables of the system. The simulation should also lend itself well to a distributed computing environment. The numerous design tasks of a large system design should be computationally distributed among the participating design groups. The simulation should be *robust*; it should accommodate a wide range of system sizes and complexities. Finally, the simulation here is *random* in that it creates design scenarios that may have been initially unforeseen by the system analysts. To this end, the author has designed a computer program, coded in FORTRAN, and called CASCADE (Complex Application Simulator for the Creation of Analytical Design Equations).

CASCADE accepts user inputs to randomly construct and then converge a large system of complex equations. This system of equations should be viewed as an analytical representation to a real-life design scenario. After the user tells CASCADE the desired size of the system, the *nature* (the initial coefficient, the sign, the coupling, and the exponent) of each term, of each equation, of each subsystem of the system is determined randomly, using a random number generator. The non-hierarchic system is constructed on the first iteration, and converged on iterations thereafter, after having initialized each and every output equation to a value of unity. This procedure was first implemented on a single computer format, cycling though all of the subsystems in the system, one-by-one, in a sequential manner. The procedure was later modified for solution in a distributed computing environment, using Parallel Virtual Machine (PVM). A virtual machine is constructed, consisting of numerous local workstations. Each subsystem in the constructed system is then sent to a separate computer for analysis, such that the number of subsystems being analyzed at one time is equal to the number of computers on the virtual machine. Inherently, this appears to be more efficient, and more realistic.

Once the system is constructed and converged, either in a single computer or distributed computing manner, CASCADE offers numerous post-convergence features. The Global Sensitivity Equation method can be used to compute the total sensitivities of the system outputs, with respect to the inputs. This is done by first computing sensitivities on a local level. The matrices from which the total derivatives are computed can either be normalized or not. Normalized matrices offer a higher likelihood of an accurate solution. A second important feature is the option to write each equation of the converged system to a separate subroutine. This could be beneficial to sequencing researchers. Again, these researchers might perturb the design variables of the converged system, and then analyze the various ordering possibilities of the subroutines to see which sequence would attain a new converged solution most quickly. A final important feature is the option to write the converged system to a *parameters* file. This file provides a comprehensive listing of the nature of each term in the system that has been constructed, as well as other system statistics.

The previous section analyzed the results of numerous executions of CASCADE on a global level. The single computer results saw an increase in CPU time with an increase in system size. As expected, systems that were solved using the Moriarty supercomputer solved much faster than those solved on the local EAS00 workstation. Unfortunately, the unaltered PVM results did not live up to their high expectations. The CPU times for the parallel machine-generated systems were larger than those for the single

computer systems. Moreover, CPU times per iteration were larger, when a *larger* number of slave computers were used on the virtual machine. It is probable that the time required to pass information from the master machine to the slave machines is what dominated the convergence time for the PVM scenarios. This information that was passed included large, statically dimensioned, multi-dimensional arrays, that could only be sent from machine to machine *matrix element by matrix element*.

To reduce the impact of message passing, sleep times were introduced into the convergence procedure. On both the single computer and distributed computing scenarios, each subsystem analysis was performed, and then followed by a user specified period of sleep. At 0.01 seconds of sleep time, the advantages of distributed computing were first detected. At 0.05 seconds of sleep time, the use of distributed computing becomes more advantageous than the use of the Moriarty supercomputer for building and converging systems, for *all system sizes*. With 0.5 seconds of sleep time, the advantages of using a larger number of slave machines for distributed computing start to become evident. The bottom line is that the CASCADE-generated systems, complex as they are, are not by themselves complex enough to exploit the benefits of distributed processing.

Matrix normalization was found to be beneficial, for the computation of the total derivative matrix by using the GSE method. The condition number of the solution matrix was *lower* when using normalization techniques. This is an indication of a more reliable numerical estimate. The CPU times required to attain the normalized derivative matrix were also lower, than those required to attain not-normalized derivative matrices.

Future Work

The first step that should be taken for the advancement of this research would be to improve the results associated with distributed computing and PVM. A method must be found to more effectively pass the large array data from the master to the slave. Message passing was found to be the dominating factor in the time taken to build and converge the systems, while using distributed processing techniques. An interface is required that will enable the PVM message passing to bypass select network layers and avoid performance degradation due to communication through the operating system and a transport layer protocol (as in Figure 10).

This *desired interface* allows PVM to execute the application (CASCADE), and bypass the protocol overhead of UDP and IP, and communicate directly to the low-level network interface. For this preliminary work, the low-level network has been Ethernet-based. Eventually, however, a conversion to an ATM-based network is foreseen. Further time-related improvement could be made by setting up a virtual machine whose hosts consist of the numerous processors on a multi-processor supercomputer.

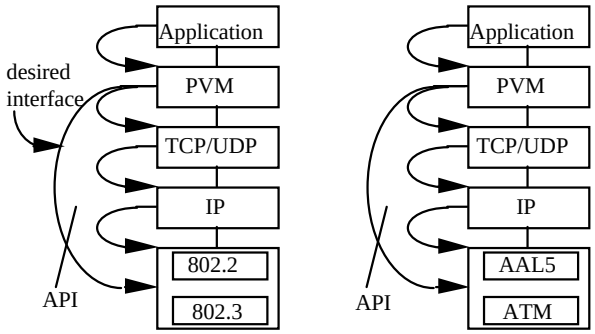

Figure 10. Direct message passing to Network layer.

The size of systems created by CASCADE has been limited by the static dimensioning allowances of the FORTRAN programming language. If this were overcome, systems of infinite size could potentially be constructed and converged. This would prove that the principals of MDO could be extended to systems of any imaginable size, with couplings having any imaginable complexity.

A final concept is to physically combine the CASCADE package with the other MDO-related program methodologies that have been discussed, such as task sequencing software and coupling suspension software. An MDO-framework is envisioned, in which a complex system could be randomly built and converged, then optimally altered and re-converged. This framework could make use of Virtual Reality techniques, to provide the user with a 3-dimensional representation of a design, and virtual foresight to the effects that result from and hypothetical changes made to the design.

Acknowledgments

The authors wish to acknowledge partial support of this work under grant 150-82350 of the UB Multidisciplinary Seed Pilot Project Program and NSF PFF Grant DMI 9553210.

References

[1] Bloebaum, Christina L., "Global Sensitivity Analysis in Control-Augmented Structural Synthesis," AIAA Student Journal, Summer 1989.

[2] Bloebaum, Christina L., "Formal and Heuristic System Decomposition Methods in Multidisciplinary Synthesis," Ph.D. Thesis, University of Florida, Gainsville, FL, 1991.

[3] Bloebaum, C. L., "An Intelligent Decomposition Approach for Coupled Engineering Systems". Fourth AIAA/USAF/NASA/OAI Symposium on Multidisciplinary Analysis and Optimization, Cleveland, OH, September, 1992.

[4] Cheney, Ward, and Kincaid, David, *Numerical mathematics and computing*, Brooks/Cole Publishing company, Pacific Grove, California, 1985.

[5] Codenotti, Bruno and Leoncini, Mauro, *Introduction to Parallel Processing*, Addison-Wesley, Wokingham, England, 1992.

[6] Eason, Ernest D. and Wright, Joyce E., "Implementation of Non-Hierarchic Decomposition for Multidisciplinary System Optimization." Fourth AIAA/USAF/NASA/OAI Symposium on Multidisciplinary Analysis and Optimization, Cleveland, OH, September, 1992.

[7] Eschenauer, Hans A. and Weinert, Matthias, "Approximation Concepts for the Decomposition-Based Optimization of Complex Mechanical Structures on Parallel Computers," Advances in Design Automation Vol. II, ASME Pub. DE-Vol. 65-2, 337-345, 1993.

[8] Flannery, Brian P., Teukolsky, Saul A., and Vetterling, William T., *Numerical Recipes - the art of scientific computing*, Cambridge University Press, Cambridge, 1986.

[9] Geist, A., Beguelin, A., Dongerra, J., Weicheng, J., Mancheck, R., and Sunderam, V., *PVM: Parallel Virtual Machine - A user's guide and tutorial for networked parallel computing*, The MIT Press, Cambridge, Massachussets, 1994.

[10] Hajela, P., Bloebaum, Christina L., and Sobieszczanski-Sobieski, Jaroslaw, "Application of Global Sensitivity Equations in Multidisciplinary Aircraft Synthesis," Journal of Aircraft, Volume 27, No. 12, 1990, pp. 1002 - 1010.

[11] Huddleston, John V., *Introduction to Computers, FORTRAN version*, Exchange Publishing Division, Buffalo, NY, 1988.

[12] Hulme, Kevin F., "Development of CASCADE - A Multidisciplinary Design Optimization Test Simulator for use in Distributed Computing Environments", Masters Thesis, University of Buffalo, Buffalo, NY, 1996.

[13] Lakshminarayan, Krishnan, "ATM Networking and Multimedia - A White Paper," Sun Microsystems Computer Corporation, Revision X, August, 1993.

[14] Maliniak, L., "Teamwork is the Key to Concurrent Design," Electronic Design, January 1991, pp. 41-54.

[15] McCulley, C. and C. L. Bloebaum, "Optimal Sequencing for Complex Engineering Systems Using Genetic Algorithms." Fifth AIAA/USAF/NASA/OAI Symposium on Multidisciplinary Analysis and Optimization, Panama City, FL, September, 1994.

[16] McCulley, Collin M., "A Genetic Tool for Optimally Sequencing the Design of Complex Engineering Systems," Masters Thesis, University Of Buffalo, Buffalo, NY, 1995.

[17] Miller, Erik, "Coupling Suspension and Elimination in Multidisciplinary Design Optimization," Masters Thesis, University of Buffalo, Buffalo, NY, 1995.

[18] Rogers, J.L., "DeMAID -- A Design Manager's Aide for Intelligent Decomposition: User's Guide." NASA Technical Memorandum 101575, March, 1989.

[19] Sobieszczanski-Sobieski, Jaroslaw, "A Linear Decomposition Method for Optimization Problems - Blueprint for Development," NASA Technical Memorandum 83248, 1982.

[20] Sobieszczanski-Sobieski, Jaroslaw, "Multidisciplinary Systems Optimization by Linear Decomposition," Symposium on Recent Experiences in Multidisciplinary Analysis and Optimization, Hampton, VA, 1984.

[21] Sobieszczanski-Sobieski, Jaroslaw, "The Sensitivity of Complex, Internally Coupled Systems," AIAA Journal, Volume 28, No. 2, pp. 153-160.

[22] Sobieszczanski-Sobieski, Jaroslaw, "Optimization by Decomposition: A Step from Hierarchic to Non-Hierarchic Systems," Second NASA/Air Force Symposium on Recent Advances in Multidisciplinary Analysis and Optimization, Hampton, VA, September, 1988.

[23] Sobieszczanski-Sobieski, Jaroslaw, Bloebaum, Christina L., and Hajela, P., "Sensitivity of Control-Augmented Structure obtained by a System Decomposition Method," AIAA Journal, Vol. 29, No. 2, February, 1991.

[24] Steward, D.V., *System Analysis and Management*. Petrocelli Books, New York, 1981.

[25] Vanderplaats, G. N., *Numerical Optimization Techniques for Engineering Design: with Applications*, McGraw Hill, New York, NY, 1984.