8th AIAA/USAF/NASA/ISSMO
Symposium on Multidisciplinary
Analysis and Optimization
6-8 Sept. 2000 Long Beach, CA

A00-40080

AIAA-2000-4819

EFFICIENT OPTIMIZATION ALGORITHMS FOR PARALLEL APPLICATIONS

Gerhard Venter* **(gventer@vrand.com) and Brian Watson*** **(bwatson@vrand.com)**

Vanderplaats Research & Development, Inc., 1767 S. 8th Street, Suite 100, Colorado Springs, CO 80906

ABSTRACT

The present paper investigates parallelization of general purpose numerical optimization algorithms, where the optimization algorithm is coupled with an existing analysis program. Since these optimization algorithms may be coupled to almost any analysis, parallelization of the analysis itself is not considered. The paper considers a typical structural finite element model to investigate the parallel efficiency of a number of existing gradient-based algorithms and proposes a new algorithm for massively parallel applications. The new algorithm is based on statistical design of experiments (DOE). Finally, the paper also investigates the parallel efficiency when implementing these algorithms on different parallel architectures. For the existing gradient-based algorithms considered, the sequential linear programming (SLP) algorithm had the highest parallel efficiency. Initial results for the DOE based algorithm seems promising, especially when coupled with a parallel gradient-based algorithm. Finally, our investigation indicates that a shared memory architecture may not be the best choice for parallel optimization using numerical simulations with significant amounts of disk I/O.

INTRODUCTION

Despite many years of research, resulting in the availability of several general purpose optimization programs, optimization has only realized limited success in the industrial environment. There are many reasons for this lack of acceptance, including (a) lack of user familiarity with optimization concepts, and (b) immense computational resource requirements for general purpose optimization. The first issue is due to the fact that optimization is rarely taught at the undergraduate level, creating the need for user training, that companies are often unwilling to invest in. The VisualDOC[1] program, from VR&D, was created to address this issue. It provides an intuitive graphical user interface, guiding the user through the steps required to set up an optimization problem. This environment allows engineers to apply optimization to real problems with minimal training. The high computational resource requirement of general purpose optimization stems from the general purpose nature of the problem and is the focus of this paper.

Engineers typically use tools that were developed for performing only a single analysis. To perform optimization using these tools, a large number of analyses are required, either to provide gradient information via finite difference calculations, or to provide data for response surface or other non-gradient based optimization methods. A typical industrial analysis can require many hours of computer time. Given the time constraints that are placed on design engineers, this makes many potential optimization problems impractical. Parallel processing has the potential to reduce the time requirements such that general purpose optimization becomes practical for a wide range of industrial applications.

Most general purpose optimization techniques are gradient-based. In general, gradient-based optimization algorithms reach an optimum design point by moving from one design point to the next. This process of moving from one design point to the next typically consists of calculating the gradient values of the objective function and active constraint set to obtain a search direction, followed by a one-dimensional search in that search direction. The one-dimensional search determines how far to move in the search direction and identifies the next design point where gradient calculations will be performed. Because typical analysis packages do not generally provide gradients, most general purpose optimizers employ finite difference gradient calculations to obtain the gradients, and VisualDOC is no exception. The finite difference gradient calculations typically dominate the computational time required to complete the optimization, are independent and easily parallelized. The authors[2] first investigated the parallelization of the finite difference calculations in existing VisualDOC algorithms. Several other studies (e.g., Rogers[3], Sikiotis[4], El-Sayed[5] and Watson[6]) have also focused on the parallelization of the gradient calculations, with moderate success. The present paper provides an overview of previous results obtained by

* *Senior Research and Development Engineer, Member AIAA*

the authors related to the parallelization of existing gradient-based algorithms and the potential efficiency gain that may be obtained from these algorithms. New results obtained from an improvement implementation of the parallel SLP algorithm, and results using a 512 processor Origin 2000 (NASA Ames Research Center) are also provided.

Unless there are also a large number of independent design variables, the efficiency gain from parallelizing the finite difference gradient calculations is limited, especially in a massively parallel environment with potentially 1,000's of available processors. For forward difference calculations, one can use a maximum number of processors equal to the number of independent design variables. For central difference calculations, the number of parallel processors that can be utilized is equal to two times the number of independent design variables. Due to these limitations, it is necessary to consider radically new optimization algorithms for massively parallel applications. The present paper makes some initial progress in this direction, by proposing a new DOE-based algorithm, specifically designed for massively parallel applications.

PARALLEL IMPLEMENTATION

All algorithms discussed in the present paper implemented a master-slave paradigm. When multiple analyses are required, the master process allocates the tasks to all available slave processors. When a slave finishes its task, it again becomes available, and can be allocated another task (e.g., Smith[7]). This paradigm is ideally suited to a heterogeneous parallel environment, such as a local area network of workstations, because it is intrinsically dynamically load balanced. That is, faster processors will be allocated more tasks. Also, this scheme requires only minimal inter-processor communication. The design variable values are sent to the slaves, and the response values are sent back to the master.

The results presented in the present paper were obtained from two systems, in both cases using the message passing interface (MPI) standard for message passing. The first system was a heterogeneous cluster of existing UNIX and Windows NT workstations, available at VR&D. These machines were connected using a local area network and we used the MPI implementation contained in the local area multiprocessor (LAM) software originally developed by Ohio State University. The second system was a shared memory, 512 processor Origin 2000 available from NASA Ames Research Center. Throughout the paper it is clearly stated what system, and how many processors were used to generate results.

EXAMPLE PROBLEM

Structural optimization of a typical aircraft wing was considered as an example problem for testing and evaluating parallel algorithms. The wing structure considered is constructed of aluminum and has a length of 70 *ft*. A finite element model of the wing was constructed to evaluate the required stresses, displacements and frequency constraints. The linear finite element analyses required during the optimization process were performed using GENESIS[8]. The finite element model consisted of 2,400 two-dimensional shell elements (CQUAD4), 600 one-dimensional truss elements (CROD) and has a total of 1,917 nodes. This finite element model is shown graphically in Fig. 1.

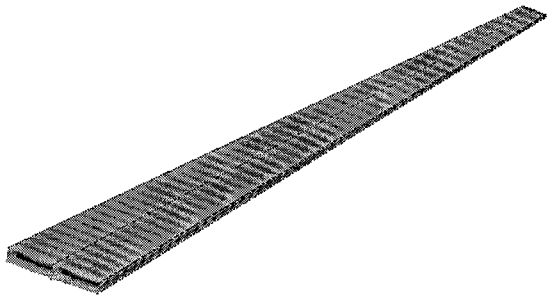

Figure 1: Finite element model for the wing example problem.

Three load conditions, typical of an aircraft wing, are considered during the optimization as follows:

- **Load Case 1 (Static):** Normal lift and engine weight
- **Load Case 2 (Static):** Landing, half lift and engine weight
- **Load Case 3 (Frequency):** Fundamental frequency

The optimization problem is defined as minimizing the mass of the wing with the three load cases applied and subject to stress, displacement and natural frequency constraints, resulting in a total of 21,648 constraints. The problem has a total of 125 design variables. One hundred of these design variables represent thickness values of groups of shell elements with lower bounds of 0.02 *in*, upper bounds of 1.00 *in* and initial values of 0.2 *in*. The remaining twenty-five design variables represent the cross-sectional area of groups of rod elements, with lower bounds of 0.5 *in*, upper bounds of 2.0 *in* and initial values of 1.0 *in*.

The GENESIS software was used to perform the finite element analyses (i.e., function evaluations) required for all the parallel optimization algorithms

investigated here. However, although GENESIS provides excellent finite element analysis capabilities, it is primarily a powerful *structural optimization* tool. To obtain a baseline optimum design for validating our parallel results, the wing structure was optimized using GENESIS. Note that using the optimum result obtained by GENESIS we may consider any number of design variables up 125 and still compare our results with that found by GENESIS. For example if we consider ten design variables, we may use the initial values for these ten design variables while setting the values of the remaining design variables to the optimum values obtained from GENESIS.

EXPLOITING PARALLELISM IN EXISTING ALGORITHMS

The computational time for gradient-based, general purpose optimization algorithms may be divided into three main parts:

- Analyses required for gradients
- Analyses required for one-dimensional search
- Other optimization computations

For problems with moderate numbers of design variables, the time to complete the analyses, both for gradient and one-dimensional search calculations, dominates the total solution time. Additionally, the number of one-dimensional search calculations is fairly independent of the number of design variables. However, the number of analyses required to perform the finite difference gradient calculations increases with the number of design variables and will dominate the total solution time for any problem with more than just a few design variables.

This section summarizes some initial results obtained by parallelization of the finite difference gradient calculations of current VisualDOC algorithms. These include the Modified Method of Feasible Directions (MMFD), Sequential Linear Programming (SLP) and Sequential Quadratic Programming (SQP). Next, an improved implementation for the parallel SLP algorithm is introduced, and finally the scalability of the SLP algorithm on a 512 processor shared memory Origin 2000 is investigated.

Initial results for existing algorithms

The authors investigate the effectiveness of parallelizing only the finite difference gradient calculations (Ref. 2) of the existing optimization algorithms within VisualDOC. This is mainly due to the fact that the finite difference gradient calculations are independent and are easily parallelized, while the one-dimensional search is inherently an iterative process that is difficult to parallelize. A function, F that depends on a single variable, x, may be used to illustrate how the finite difference gradient calculations are performed in parallel. The gradient of F with respect to x at the point x_0 is defined as

$$\left.\frac{dF(x)}{dx}\right|_{x=x_0} = \lim_{x \to x_0} \frac{F(x) - F(x_0)}{x - x_0} \qquad (1)$$

The gradient of F with respect to x may be approximated at the point x_0, by perturbing x with a small value, h, as follows:

$$\left.\frac{dF(x)}{dx}\right|_{x=x_0} \approx \frac{F(x_0 + h) - F(x_0)}{h} \; . \qquad (2)$$

The right hand side of Eq. (2) is the forward finite difference formula for the gradient of F with respect to x at $x=x_0$. For the case where F is a function of many variables, the right hand side of Eq. (2) is repeated for each variable, resulting in a number of function evaluations that are independent of each other. It is thus fairly easy to parallelize the finite difference gradient calculations, in which case a number of processors equal to the number of design variables may be used.

In Ref. 2 the authors considered the wing optimization problem introduced in the previous section. The problem was investigated first using twelve and then using twenty-four design variables. In each of the two cases, the MMFD, SLP and SQP algorithms of VisualDOC were used to perform the optimization, thus studying both the effects of the problem size and optimization algorithm on the overall parallel efficiency. The parallel runs were performed on a cluster of six workstations consisting of SUN, SGI and Windows NT machines. In all cases the optimum results found by the parallel algorithms showed excellent correlation with the optimum results obtained from GENESIS.

The results obtained indicate that for the serial implementation, the MMFD algorithm was the most efficient in solving the example problem. However, in the parallel implementation the SLP algorithm was most efficient, even though it required significantly more analyses. The reason for the higher parallel efficiency of the SLP algorithm is the fact that it does not have one-dimensional search calculations. Additionally, the results show that the number of one-dimensional search calculations remain fairly constant as the number of design variables increases, thus increasing the parallel efficiency of all the algorithms for larger problem sizes.

In all cases, performing the gradient calculations in parallel resulted in a significant increase in performance, as summarized in Table 1. In Table 1, the slowest serial time represents a worst case scenario and

indicates the time it would take to perform the optimization in series using the slowest processor of the virtual parallel machine. In contrast, the fastest serial time indicates the time it would take to perform the optimization in series using the fastest processor of the virtual parallel machine.

Improved parallel SLP algorithm

As illustrated in the previous section, the SLP algorithm had the highest parallel efficiency of the current VisualDOC algorithms. However, since publishing our initial results, we realized that the parallel efficiency of the SLP algorithm could be greatly improved. The initial parallel SLP implementation considered each design iteration as a two-step process. First, the current design point is evaluated in a serial manner, using a single processor. Next, the finite difference gradient calculations at the current design point are performed in parallel. However, it is very easy to evaluate both the current design point and the finite difference gradient calculations at that point in a single step, all done in parallel. In the ideal case, where one has a number of processors equal to the number of design variables plus one, the improved SLP implementation can reduce the overall time required to complete an optimization by a factor of 2.

However, the SLP algorithm implemented in VisualDOC does not always calculate gradients at each design iteration. If the new design iteration results in a worse objective or larger constraint violations, VisualDOC automatically reduces the move limits and does a new function evaluation in the same search direction, without evaluating any gradients. Due to this process of reducing the move limits without re-evaluating the gradients, the maximum speedup of 2 is not always realized. However, the speedup associated with using the new implementation is generally still significant.

To test the increase in parallel efficiency, we applied both the old and the new SLP implementations to the wing problem with twelve design variables. This problem was run on the 512 processor Origin 2000, using thirteen processors and the timing results are summarized in Fig. 2. For this example, the original implementation required 43.72 minutes to complete the optimization, while the new implementation required only 24.33 minutes, a speedup of 1.8. As mentioned, the maximum speedup of 2 is not always realized, but in general the speedup is still significant.

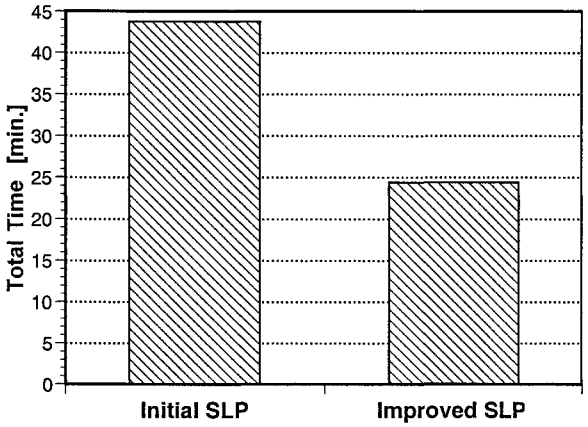

Figure 2: Total time for the 12 design variable wing problem, using 13 processors.

Large scale application

Using our initial results as a basis, we wanted to investigate the scalability of our implementation to large numbers of processors. We gained access to a 512 processor Origin 2000 at NASA Ames Research Center and decided to port and test our algorithms on this machine. Since the SLP algorithm had the highest parallel efficiency, even with the original implementation, we decided to concentrate our efforts on this algorithm.

The Origin 2000 is a shared memory machine and the 512 processor machine at NASA Ames Research Center is currently the largest of its type in the world. We used the new SLP implementation and considered several different cases where the number of design variables was equal to 12, 24, 75 and 125. In each case we used a number of processors equal to the number of design variables plus one. The results are summarized in Fig. 3. Due to small variations in the number of

Table 1: Total time to complete the twelve design variable case.

	Twelve Design Variables			Twenty Four Design Variables		
	MMFD	**SLP**	**SQP**	**MMFD**	**SLP**	**SQP**
Parallel Time [s]	7,479	6,753	5,964	12,300	8,754	9,476
Slowest Serial Time [s]	31,020 (4.15)	52,580 (7.79)	36,080 (6.05)	56,100 (4.56)	72,380 (8.27)	70,620 (7.45)
Fastest Serial Time [s]	12,267 (1.64)	20,793 (3.08)	14,268 (2.39)	22,185 (1.80)	28,623 (3.27)	27,927 (2.95)

(Values in parentheses are the speedup factor between the parallel and serial optimizations)

design iterations required to solve each problem, Fig. 3 shows the average time per design iteration as a function of the number of processors for each problem.

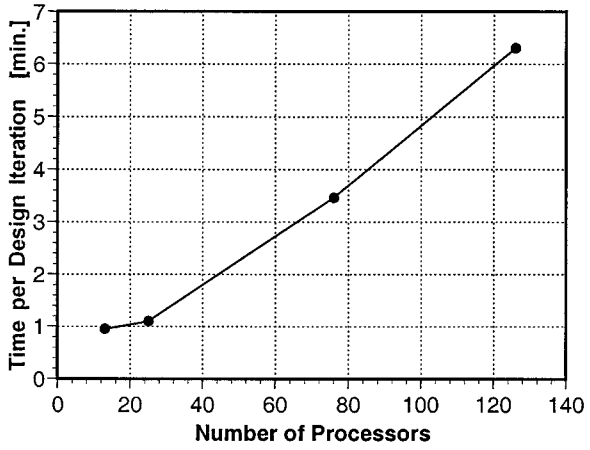

Figure 3: Average time per design iteration as a function of the number o processors for the wing problem (number of processors is equal to the number of design variables plus one).

The results of Fig. 3 were unexpected, instead if being independent on the number of design variables (and thus the number of processors), the overall time increased with an increase in the number of design variables.

We decided to further investigate the results of Fig. 3, concentrating on the 75 and 125 design variable cases. For each of these two cases, we performed the optimization using different numbers of processors. For the 75 design variable case, we used 19, 38 and 76 processors and for the 125 design variable case we used 16, 32, 63 and 126 processors. We found that, except for a small number of processors (less than roughly 30 to 40), increasing the number of processors also increased the overall solution time. This is clearly illustrated if we calculate the average time required to evaluate a set of parallel computations. For this calculation, a set of parallel computations represents a number of function evaluations equal to the number of processors used. In the ideal case, this time should be constant and equal to the time of a single analysis performed on a single processor. We calculated the average time, using the following equation:

$$T_{avg} = \frac{T_{total} N_{proc}}{N_{iter}(N_{dvar} + 1)} \qquad (3)$$

where T_{avg} denotes the average time required to evaluate a set of parallel computations and T_{total} denotes the total time required to complete the optimization. Additionally, N_{proc} denotes the number of processors used in the optimization, N_{iter} denotes the number of design iterations required to complete the optimization and N_{dvar} denotes the number of design variables of the optimization problem. The results of all previous cases (different number of design variables and processors) are shown in Fig. 4.

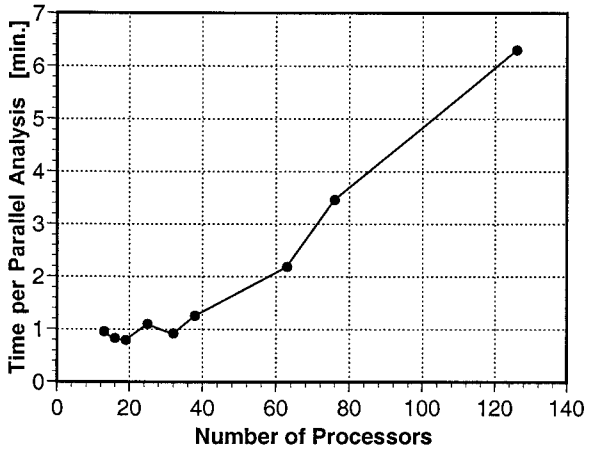

Figure 4: Average time per set of parallel calculations.

Figure 4 clearly illustrates that the average time required to evaluate a set of parallel computations greatly increases with an increase in the number of processors. In explaining these results, remember that we implemented a master-slave paradigm with minimal inter-processor communication. In our implementation the master processor sends the current design variable values to the slave processors and the slave processors send back the response values to the master. As a result of our implementation, we do not consider inter-processor communication as the problem. However, GENESIS does generate a fairly large amount of output that is stored on the hard disk. It is our inclination that it is this disk I/O that creates a bottleneck and deteriorates the parallel efficiency of our algorithms when using larger number of processors.

Discussion

Even though the results showed a significant decrease in the overall time required to perform an optimization, there is one major drawback when parallelizing existing gradient-based algorithms in a massively parallel environment: The number of processors that may be used by the gradient calculations is limited by the number of independent design variables in the problem. For forward finite difference calculations, the maximum number of processors that can be used is equal to the number of independent design variables, while for central difference calculations it is equal to twice the number of independent design variables. This limitation on the number of processors that can be used by parallelizing existing algorithms makes the approach more useful for distributed computing on an existing

network of workstations than for massively parallel applications.

Additionally, for our course-grained parallel implementation, the shared memory Origin 2000 may not be the best platform to achieve optimal performance when using large numbers of processors. With the little inter-processor communication and substantial disk I/O associated with our implementation, it may be better to utilize a cluster of workstations, each with its own processor, memory and hard disk.

NEW ALGORITHM FOR MASSIVELY PARALLEL APPLICATIONS

Based on the results of the investigation summarized in the previous section, it is concluded that new optimization algorithms specifically designed for a massively parallel environment are required to take full advantage of the parallel efficiency within such an environment. When considering these new algorithms it is important to note that the most efficient algorithm for a non-parallel application is not necessarily the most efficient in a parallel environment. This is clearly illustrated in the previous section, when comparing the MMFD and SLP algorithms. In considering an algorithm for massively parallel applications, the total number of analyses is less important than the efficiency of the algorithm in using large numbers of processors simultaneously.

DOE-based massively parallel algorithm

We propose a new design of experiments (DOE) based parallel optimization algorithm for application with massively parallel machines. The basic algorithm may be summarized as:

1. Use DOE to obtain a set of design points that are spread throughout the current design space.
2. Evaluate all points identified by the DOE in parallel and keep track of the best design point.
3. Center and optionally change the size of the design space about the best point found in Step 2.
4. Make sure the new design space does not violate any of the original side constraints by adjusting the bounds if necessary.
5. Go back to step 1 and repeat until convergence.

If enough processors are available, the time required to complete each design iteration is equal to the time of a single analysis. To reduce the overall time required to complete the optimization, the goal is to perform the smallest number of required design iterations.

Dealing with unconstrained optimization problems using the current algorithm is straightforward. To deal with constrained optimization problems, we need to take account of the constraints in defining the best design point. In general, the algorithm has two phases with different priorities. The first phase occurs when the best design point is infeasible. In this phase, the algorithm's priority is to find a feasible design, irrespective of the objective function value. The second phase occurs once a feasible design point has been found. In this phase, the algorithm's priority is to improve the objective function value, irrespective of the constraint margin. Note that once a feasible point is found, no more infeasible design points are considered in determining the best design.

One of the most important factors influencing the efficiency and robustness of the proposed DOE based algorithm, is the windowing or move limit strategy used during the optimization process. The goal is to identify a small region, referred to as a window, about the optimum solution within the original design space. The windowing strategy used here always centers each new window about the best design point found in previous design iterations. Note that the best design point is saved from one design iteration to the next and is only replaced if a better design is found. Additionally, the window size is potentially adjusted for each new iteration.

First, a factor is defined for reducing the window size from one iteration to the next. This factor is adjusted at the end of each design iteration by multiplying the current factor with a constant number greater than one. The effect is a more aggressive reduction in window size as the optimization progresses. The factor for reducing the window size is not directly applied to the current window. Instead, a linear interpolation is used to calculate the actual factor used for reducing the window size, as follows:

$$\alpha_i = \frac{|C_i - B_i|}{C_i - L_i}$$

$$Adj_i = \alpha_i + (1 - \alpha_i) OrigAdj \qquad (4)$$

Equation 4 is applied on a per design variable basis. In Eq. 4, C_i denotes the center point, B_i denotes the best point and L_i denotes the lower bound for design variable i. $OrigAdj$ denotes the original factor for reducing the window size (this is adjusted after each design iteration as discussed above) and Adj_i denotes the actual factor that will be used for design variable i. The result of this linear interpolation is that the window size will not be reduced when the best point is located

on a boundary, while it will be reduced with a factor equal to *OrigAdj* when the best point is located at the center.

Additionally, a history of the location of the best design point with respect to the bounds of the current window is kept. Again this is done on a per design variable basis. If the best design point hits the same bound twice, a fixed factor is used to enlarge the window size for that design variable. However, if a design variable oscillates between its upper and lower bounds, the window size for that design variable is reduced by a fixed factor.

Another important aspect of the proposed algorithm is the DOE used to identify design points during the search. Most statistical DOE algorithms minimize some measure of variance associated with an assumed mathematical model that describes the underlying response. The result is a set of design points that are located on the boundary of the design space. In our case, a non-traditional design that is independent of the underlying mathematical model and that spreads the design points throughout the design space (also referred to as a space filling design) is required. Two such designs are the modified maximum distance design of Audze and Eglais[9] and orthogonal arrays. Owen[10] implemented a series of orthogonal arrays in freely available software that is ideally suited for our purpose and was implemented in the example problem.

Results

For the example problem, we used an orthogonal array design of experiments, using the software by Owen[10]. This DOE resulted in a design with 50 design points for the 10 design variable problem considered here. Each design variable takes on 5 different values between its upper and lower bounds.

The DOE-based algorithm used a factor of 3.0 for reducing the window size form one iteration to the next. This value is adjusted by increasing its value with 10% each design cycle is completed. Additionally, the fixed factors for enlarging the window size when the same bound is hit more than once and for reducing the window size in the cases of oscillation between the upper and lower bounds were both taken as 2. Finally, the algorithm was setup to complete ten design cycles.

The objective function history as a function of design iterations is shown in Fig. 5. Although some design points considered during the optimization were infeasible, the best design for each design cycle shown in Fig. 5 are feasible. Figure 5 illustrates that the proposed algorithm was able to significantly reduce the objective function within the first three design iterations. However, after the 3rd design iteration the algorithm was slow to improve the best design point.

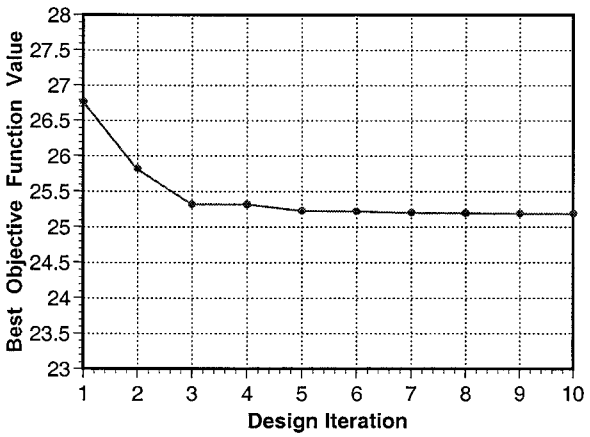

Figure 5: Objective function history for the DOE based algorithm

The best design point found after ten iterations is summarized in Table 2. Table 2 also contains the optimum results found by GENESIS. Note that the best objective function value found by the DOE based algorithm is only 1.7% different from that found by GENESIS.

Table 2: Optimum results found by GENESIS and by the DOE based algorithm

Parameter	GENESIS Results	DOE Based Results
Mass	24.745	25.177
Thickness 1	0.41255	0.42051
Thickness 2	0.43578	0.41161
Thickness 3	0.43426	0.55898
Thickness 4	0.42493	0.49154
Thickness 5	0.43661	0.44099
Thickness 6	0.43313	0.37624
Thickness 7	0.42713	0.63151
Thickness 8	0.42077	0.50812
Thickness 9	0.40779	0.25192
Thickness 10	0.39425	0.30998

Improvement DOE-based algorithm and future work

The proposed algorithm is efficient in narrowing the design space about the optimum solution. However, the algorithm is slow in converging on a final optimum solution. This raises an interesting point of how to exploit the advantages of this new algorithm while avoiding its weaknesses. We decided to investigate an alternative two-step process to enhance the convergence characteristics of the DOE-based algorithm. An example of such a two-step process would be to first apply

the DOE-based algorithm for a few design iterations to explore the design space and identify a reasonably good design point. The next step is to use the design point identified by the DOE-based algorithm as a starting point for a parallel gradient-based algorithm. The idea is to start the gradient-based algorithm close to an optimum solution and hence improve the convergence characteristics of the DOE-base algorithm. Because of the high parallel efficiency of the improved parallel SLP algorithm, we decided to use this algorithm in the second step.

We used the best design point after the third design iteration of the DOE-based optimization (Fig. 5) as a starting point for the SLP algorithm. Note, that we used the original bounds for the design variables and only changed the starting point. From this starting point, the SLP converged in only 4 design iterations. Remember that if one has enough processors, each design iteration of both the SLP and the DOE-based algorithms can be performed in the equivalent time of a single function evaluation. This means that if one has 50 processors, this 10 design variable problem could be solved in the equivalent time of only 7 function evaluations. In contrast, using the parallel SLP algorithm starting from the original staring point would require 24 design iterations.

Discussion

Based on our initial investigation, the proposed DOE-based algorithm is capable of efficiently using large numbers of processors in systematically exploring the design space. However, the DOE-based algorithm is slow to converge on the optimum design point. Combining the DOE-based algorithm with and the improved parallel SLP algorithm seems to be a promising alternative. This two-step process is still not fully explored and future work would have to investigate when to switch from the DOE-based algorithm to the SLP algorithm. Additionally, the DOE-based algorithm does not only provide a reasonable starting point for the SLP algorithm, but also reduces the bounds about this point. Future work will have to investigate the most efficient use of these reduced bounds in the SLP algorithm.

CONCLUSION

Our results showed a significant decrease in the overall time required to perform general-purpose optimization using existing gradient-based algorithms with the finite difference gradient calculations performed in parallel. We found that the SLP algorithm had the highest parallel efficiency among the algorithms that we considered. If one has as many processors as the number of design variables plus one, each design iteration may be performed in the time of a single function evaluation. This high parallel efficiency of the SLP algorithm is due to the fact that no one-dimensional search calculations are required.

Our results also indicate that when using large number of processors and a numerical simulation that produces large amounts of disk I/O as an analysis, a shared memory machine like the Origin 2000 may not be the best choice. For the course grained implementation used here, with little inter-processor communication and substantial disk I/O, it may be better to utilize a cluster of workstations, each with its own processor, memory and hard disk.

The paper also points out that the gain in efficiency by parallelizing existing gradient-based algorithms is limited by the number of independent design variables of the problem under consideration. For large numbers of processors new algorithms are required. In these algorithms the efficient use of large numbers of processors is more important that the total number of function evaluations. We introduced a DOE-based algorithm that, especially when used in a two-step process with a parallel gradient-based algorithm, gave very encouraging results. Using the two-step approach and 50 processors, we were able to solve a ten design variable problem in the equivalent time of 7 function evaluations. However, this DOE-based algorithm is just an initial proposal and additional future work is required.

ACKNOWLEDGEMENT

This work was sponsored in part by NASA Contract NAS1-98151.

REFERENCES

[1] VisualDOC Design Optimization Software, Version 1.0 Reference Manual, Vanderplaats Research & Development, Inc., Colorado Springs, CO, 1998.

[2] Venter, G, and Watson, B.C., "Exploiting Parallelism in General Purpose Optimization", Proceedings of the 6th International Conference on Applications of High-Performance Computers in Engineering, Maui, Hawaii, January 26-28, 2000.

[3] Rogers, J.L., Young, K.C. and Barthelemy, J.M., "Distributed Computer System Enhances Productivity for SRB Joint Optimization", 28th AIAA/ASME/ASCE/AHS Structures, Structural Dynamics and Materials Conference, Monterey, CA, pp. 596-600, April 6-8, 1987.

[4] Sikiotis, E.S., and Saouma, V.E., "Parallel Structural Optimization on a Network of Computer Workstations", *Computers & Structures*, Vol. 29, No. 1 pp. 141-150, 1988.

[5] El-Sayed, M.E.M., and Hsiung, C.K., "Design Optimization with Parallel Sensitivity Analysis on the CRAY X-MP", *Structural Optimization*, Vol. 3, pp. 247-251, 1991.

[6] Watson, B.C., and Noor, A.K., "Sensitivity Analysis for Large-Deflection and Postbuckling Responses on Distributed-Memory Computers", *Computer Methods in Applied Mechanics and Engineering*, Vol. 129, pp. 393-409, 1996.

[7] Smith, S.L., and Schnabel, R.B., "Centralized and Distributed Dynamic Scheduling for Adaptive, Parallel Algorithms", *Unstructured Scientific Computation on Scalable Multiprocessors*, eds. P. Mehrotra, J. Saltz, and R. Voigt, MIT Press, Cambridge, MA, pp.301-322, 1992.

[8] GENESIS Structural Optimization Software, Version 5.0 User Manual, VMA Engineering, Colorado Springs, CO, 1998.

[9] Audze, P. and Eglais, V., "New Approach for Planning out of Experiments," *Problems of Dynamics and Strengths*, Vol. 35, 1977, pp. 104-107, Riga, Zinatne Publishing House (in Russian).

[10] Owen, A.B., "Orthogonal Arrays for Computer Experiments, Integration and Visualization," *Statistica Sinica*, 1992, Vol. 2, pp. 439-452.