Study of a Global Design Space Exploration Method for Aerospace Vehicles

Chuck A. Baker, Layne T. Watson, Bernard Grossman, William H. Mason
Multidisciplinary Analysis and Design (MAD) Center for Advanced Vehicles
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24061-0203
540-231-9472
chuck.baker@vt.edu

Steven E. Cox, Raphael T. Haftka
Department of Aerospace Engineering, Mechanics and Engineering Science
University of Florida
Gainesville, Florida 32611-6250

Abstract

In the early stages of the design process of aerospace vehicles, the search for optimal configurations encompasses a broad range of possibilities, and the use of local optimization tools may risk missing the best designs. Therefore, global optimization methods are attractive for the early design stage. Unfortunately, global design optimization usually requires the evaluation of a very large number of designs, a formidable computational challenge. The present work uses a highly effective Lipschitzian global optimization algorithm in the multidisciplinary optimization of a High Speed Civil Transport (HSCT). The use of massively parallel computers to handle this computational challenge is also demonstrated. A variety of load balancing methods are evaluated to assess the utilization of the computer nodes.

1 Introduction

In high dimensional design spaces, the sheer volume involved inhibits the accurate characterization of the space even with an immense number of sampled points. If every vertex of a box bounding a 28 dimensional design space is sampled, 268 million points must be evaluated. Using design of experiments (DOE) theory, the number of sampled points can be significantly reduced. Even using a relatively small data set generated cheaply from DOE, there may be many important and potentially optimal regions of feasibility left uncharacterized for a complex design problem.

Previous work [1] has shown that the design space of the HSCT configuration is complex. Local minima occur in many nonconvex, disconnected feasible islands present in the design space. Running local optimizations from a sufficient number of starting points distributed throughout the design space requires a large number of function evaluations and still does not guarantee that the promising regions of the design space will be explored. A global optimizer is needed that is able to judiciously balance the local and global searches, insuring a complete space investigation, while keeping the number of function evaluations to a minimum.

Once the set of points to be evaluated is constructed, either through statistical (DOE) methods or via a direct search global optimizer, the points in the set can be evaluated concurrently. The points can easily be evaluated in parallel, but the question of how to best manage the evaluation and distribution of points on parallel computers is unresolved for exploratory multidisciplinary engineering design studies.

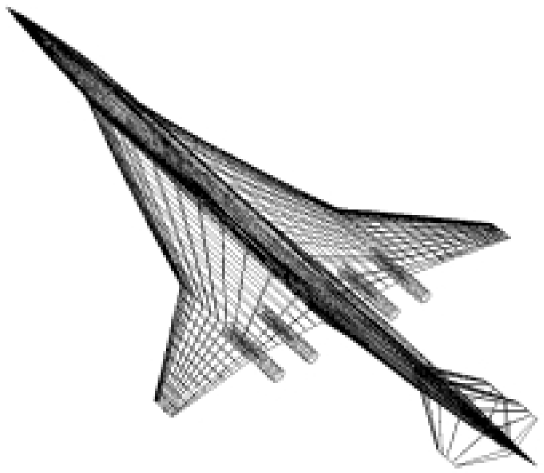

Figure 1. Typical HSCT configuration.

2 HSCT Design Problem

The design problem considered is the optimization of a HSCT configuration [10], [11] to minimize takeoff gross weight (TOGW) for a range of 5500 nautical miles and a cruise Mach number of 2.4, while carrying 251 passengers. A typical HSCT configuration is seen in Figure 1. The choice of gross weight as the objective function directly incorporates both aerodynamic and structural considerations, in that the structural design directly affects aircraft empty weight and drag, while aerodynamic performance defines the drag and thus the required fuel weight.

To successfully perform aircraft configuration optimization, it is important to have a simple, but meaningful, mathematical characterization of the geometry of the aircraft. This paper uses a model that defines the HSCT design problem using the twenty-eight design variables listed in Table 1. Twenty-four of the design variables describe the geometry of the aircraft and can be divided into six categories: wing planform, airfoil shape, tail areas, nacelle placement, and fuselage shape. In addition to the geometric parameters, four variables define the idealized cruise mission: mission fuel, engine thrust, initial cruise altitude, and constant climb rate used in the range calculation.

For the optimizer used here, upper and lower bounds have to be set on all *n* of the design variables. These bounds form a *n*-dimensional rectangular shaped set in the design space, referred to as the design *box*. In order to ensure that a thorough design space exploration was being conducted, the bounds were chosen to include as wide of a range of designs as realistically possible. The edges of the design box were set near the limits of physically impossible designs (overlapping geometries, negative chord lengths) or the assumptions of the numerical analyses being used.

Sixty-eight geometry, performance, and aerodynamic constraints, listed in Table 2, are included in the optimization. Aerodynamic and performance constraints can only be assessed after a complete analysis of the HSCT design; however, the geometric constraints can be evaluated using algebraic relations based on the 28 design variables.

Methods of varying fidelity are used for the aerodynamic and structural analyses in the constraint evaluations. The methods used to calculate the drag components used in the drag calculation and their corresponding ranges are described in [6], [7]. The aerodynamics calculations are based on the Mach box method [4], [3], and the Harris wave drag

Table 1. HSCT configuration design variables.

Index	Description
1	Wing root chord (ft)
2	Leading edge (LE) break point, x (ft)
3	LE break point, y (ft)
4	Trailing edge (TE) break point, x (ft)
5	LE wing tip, x (ft)
6	Wing tip chord (ft)
7	Wing semi-span (ft)
8	Chordwise location of max. thickness
9	LE radius parameter
10	Airfoil t/c ratio at root, (%)
11	Airfoil t/c ratio at LE break, (%)
12	Airfoil t/c ratio at LE tip, (%)
13	Fuselage restraint 1, x (ft)
14	Fuselage restraint 1, y (ft)
15	Fuselage restraint 2, x (ft)
16	Fuselage restraint 2, y (ft)
17	Fuselage restraint 3, x (ft)
18	Fuselage restraint 3, y (ft)
19	Fuselage restraint 4, x (ft)
20	Fuselage restraint 4, y (ft)
21	Nacelle 1 location (ft)
22	Nacelle 2 location (ft)
23	Vertical tail area (ft^2)
24	Horizontal tail area (ft^2)
25	Thrust per engine (lb)
26	Flight fuel (lb)
27	Starting cruise/climb altitude (ft)
28	Supersonic cruise/climb rate (ft/min)

code [5]. A simple strip boundary layer friction estimate is implemented as in [7]. A vortex lattice method with vortex lift and ground effects included [2] is used to calculate landing angle of attack. Structural weights are calculated by the FLOPS [12] weight equations. Each of these analysis methods uses iterative loops or discretization methods that can cause differences in the computational time needed to evaluate (calculate the objective function and constraint values) different HSCT designs.

In previous work [1], it was observed that multiple optima exist within the design space. A visualization technique was needed to view the topology of the design space to understand the cause of the local optima, however the dimensionality of the design prevents traditional techniques from being used. One of the design space visualization methods that was developed is shown in Figure 2. To construct this plot, three different feasible base point designs are chosen. The base points are connected to form a plane in 28 dimensional space and a grid is created in this plane. The values for the objective function and constraints are then calculated at each grid point.

Table 2. HSCT optimization constraints.

Index	Constraint
1	Fuel volume $\leq$ *50%* wing volume
2	Wing root TE $\leq$ Tail LE
3–20	Wing chord $\geq$ *7.0* ft
21	LE break within wing semi-span
22	TE break within wing semi-span
23	Root chord t/c ratio $\geq$ *1.5%*
24	LE break chord t/c ratio $\geq$ *1.5%*
25	Tip chord t/c ratio $\geq$ *1.5%*
26–30	Fuselage restraints
31	Wing spike prevention
32	Nacelle 1 inboard of nacelle 2
33	Nacelle 2 inboard of semi-span
34	Range $\geq$ *5500* nautical miles
35	C_L at landing speed $\leq$ *1*
36–53	Section C_L at landing $\leq$ *2*
54	Landing angle of attack $\leq 12^\circ$
55–58	Engine scrape at landing
59	Wing tip scrape at landing
60	TE break scrape at landing
61	Rudder deflection $\leq 22.5^\circ$
62	Bank angle at landing $\leq 5^\circ$
63	Tail deflection at approach $\leq 22.5^\circ$
64	Takeoff rotation to occur $\leq V_{min}$
65	Engine-out limit with vertical tail
66	Balanced field length $\leq$ *11000* ft
67–68	Mission segments: thrust available $\geq$ thrust required

This plot shows that even in this plane, the design space is complicated and nonconvex. The range constraint appears to be multiply connected (though it may be connected in the 28-dimensional space), providing three possible locations for local optima.

3 Lipschitzian Global Optimizer (DIRECT)

The global optimizer selected to explore the design space is a Lipschitzian unconstrained optimization algorithm that (effectively) uses all possible values of the Lipschitz constant [8]. By using different values of the constant, equal emphasis is placed on the local and global search being performed by the optimizer. This algorithm is called DIRECT because the algorithm is a direct search technique and as an acronym for *di*viding *rect*angles, one of the primary operations in the procedure.

The algorithm begins by scaling the design box to a n-dimensional unit hypercube. The center point of the hypercube is evaluated and then points are sampled at one-third the cube side length in each coordinate direction from the center point. Depending on the direction with the smallest function value, the hypercube is then subdivided into smaller rectangles, with each sampled point becoming the center of its own n-dimensional

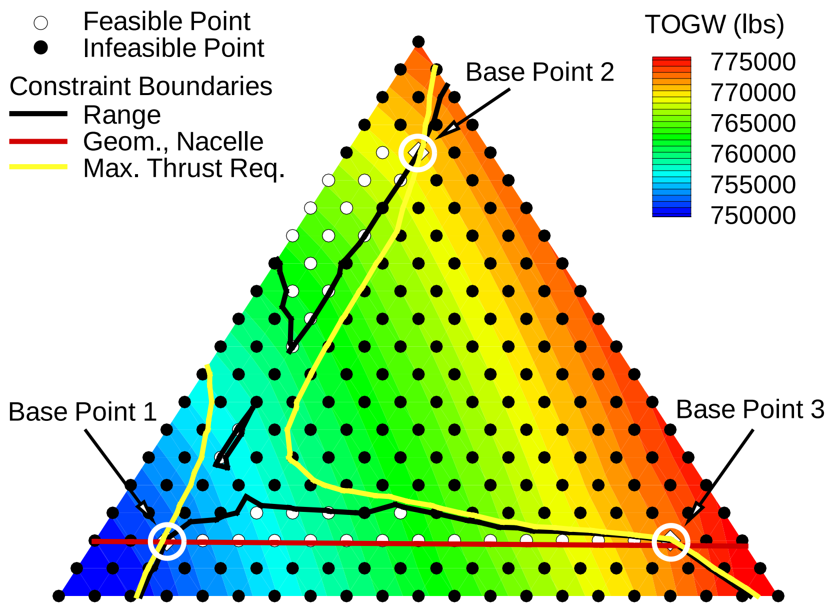

Figure 2. Design space visualization plot.

rectangle or box. All boxes are identified by their center point and their function value at that point.

From there the algorithm loops in a procedure that subdivides each of the boxes in the set in turn until termination or convergence. By using different values of the Lipschitz constant, a set of potentially optimal boxes is identified from the set of all boxes. These potentially optimal boxes are sampled in the direction of maximum side length, to prevent boxes from becoming overly skewed, and subdivided again based on the directions with the smallest function value. If the optimization continues indefinitely, all boxes will eventually be subdivided meaning that all regions of the design space will be investigated.

DIRECT's behavior on a simple 2–D test function is shown in Figure 3. The test function has local minima at (0.4,1.0), (0.9,1.0), and (0.4,0.3) with the global minimum at (0.9,0.3). The figure shows the optimizer starting from the center of the box and the division of the subsequent sub-boxes through 10 iterations. After five iterations DIRECT is beginning to converge to the local minimum at (0.4,0.3). However, due to its local–global search characteristics, by the end of 10 iterations DIRECT has refocused its search in the area of the global optimum at (0.9,0.3) where it ultimately converged.

Two important issues in using the algorithm are how to determine convergence and incorporate constraint values. For this study, the algorithm was run for a fixed number of loops or iterations. Since the purpose of the optimization was to identify promising regions of the design space, it was unnecessary to tightly converge to a global optimum. Constraints were accounted for through the use of a simple penalty function, as follows. Let x be the 28–dimensional design vector, $f(x)$ the TOGW, and $g_i(x) \leq 0$ the constraints in Table 2. The constrained optimization problem

$$\min f(x) \quad \text{subject to} \quad g_i(x) \leq 0, \; i = 1, \ldots, 68,$$

is converted to the unconstrained optimization problem

$$\min f(x) + 10 \sum_{i=1}^{68} \max\{0, g_i(x)\}.$$

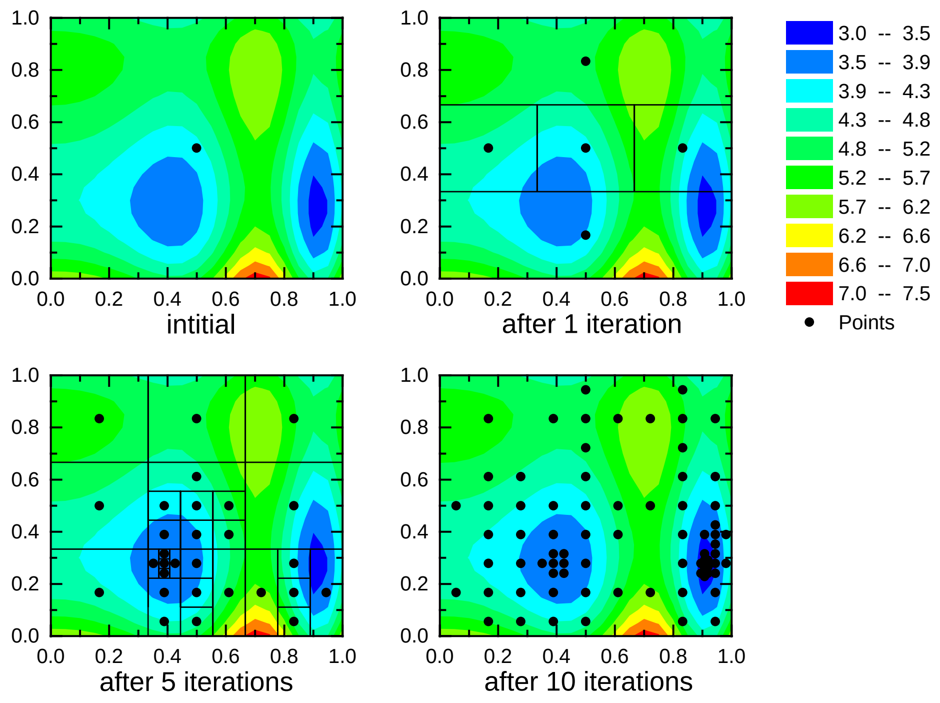

Figure 3. DIRECT behavior with test function.

4 Load Balancing Strategies

As the potentially optimal boxes are sampled in their respective directions during the DIRECT optimization, a typically large set of new design points, or tasks, that need to be evaluated is created. It is these tasks in this set of designs that are load balanced.

Processor communications were performed in the optimization algorithm through the use of the Message Passing Interface (MPI) [13], a message passing standard. MPI was chosen because, as a communications protocol, it is platform independent, thread-safe, and a widely accepted standard.

In the master-slave implementation of dynamic load balancing, one processor, the master, makes all of the calculations for box manipulation in DIRECT and controls the distribution of tasks to be evaluated by the HSCT code on the slave processors. The master processor begins with the set of all boxes, finds the potentially optimal boxes, and then samples inside of these boxes to generate the set of tasks. It then distributes one task to each slave processor. When a slave processor completes the evaluation of its task it returns the function value back to the master and receives another task, if available. The biggest potential drawback to using this method is that there is a chance for a communication bottleneck caused by slave processors simultaneously requesting work from the master.

For the static load balancing case, the processors only communicate with each other when finding the set of potentially optimal boxes and initially distributing the tasks. At the start of a DIRECT loop each processor finds its own local set of potentially optimal boxes. The processor with the highest rank gathers all of the local potentially optimal sets from the other processors and finds the global set of potentially optimal boxes. This processor creates the set of new tasks from the global set of potentially optimal boxes. The new tasks are equally distributed to all of the processors and the individual processors evaluate

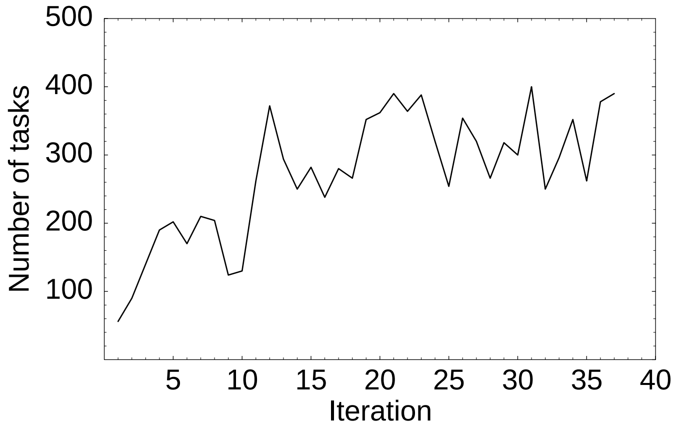

Figure 4. History of tasks per iteration.

every task in their set of new tasks. The problem inherent to static load balancing is that differences in evaluation times can cause some processors to finish their tasks early and sit idle, while other processors continue to work on their tasks.

The interprocessor communications used for the DIRECT box manipulation by the fully distributed version of dynamic load balancing are the same as those performed by the static version, with the added capability of task migration to processors that have finished their tasks. The dynamic load balancing algorithm is based on that of previous work [9], employing random polling for the redistribution of tasks and token passing to terminate the load balancing process. Once task evaluation is started by a processor, it evaluates a single task and then processes any messages received during the evaluation of the task. The cycle of evaluating and communicating is continued until the processor runs out of work, in which case it begins sending work requests to a randomly selected processor either until work is found or the termination token is received. If a work request is received by a processor, half of its remaining tasks are transferred to the requesting processor.

A dynamic load balancing strategy is also implemented that uses threads in the fully distributed version. Multi-threading in the distributed version is based on the POSIX (pthreads) package. In this implementation, one thread is a worker responsible for evaluating tasks and sitting idle when no tasks are available. A second thread handles all of the message passing and processing. By exploiting concurrency at the processor level, messages can be processed at the same time as a task is being evaluated, instead of the purely sequential operations used by the distributed version without threads.

In the subsequent discussion, these load balancing strategies are referred to as static (STATIC), dynamic load balancing with the master-slave paradigm (DLBMS), dynamic load balancing with fully distributed control (DLBDC), and dynamic load balancing with fully distributed control using pthreads (DLBDCT).

5 Optimization Results and Parallel Performance

The parallel optimization runs were conducted on an SGI Origin 2000 with a total of 64 CPUs. Runs were made on 4, 8, 16, 32, and 64 processors for each of the four load balancing methods. The DIRECT optimizer was terminated after 37 iterations, performing 10,077 function evaluations. The history of total tasks for each iteration is shown in Figure 4. The figure illustrates the amount of work that had to be distributed to the processors during the load balancing. Figure 5 is a histogram of the evaluation times for the 10,077 tasks. The variation in the evaluation times is relatively small, with most of the tasks taking around 1.75 seconds to complete.

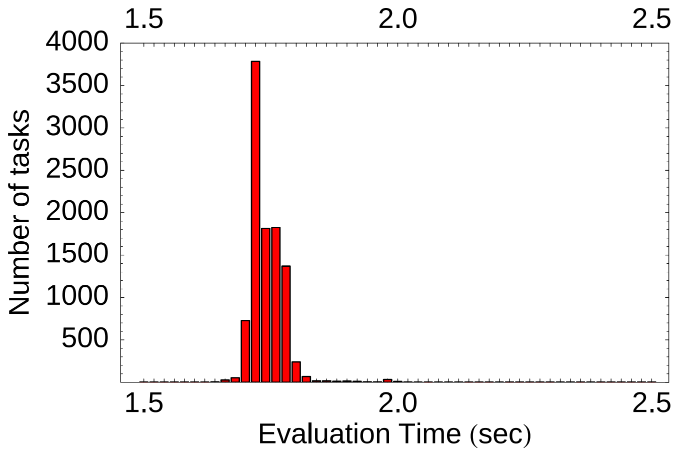

Figure 5. Time distribution for all 10,077 tasks evaluated.

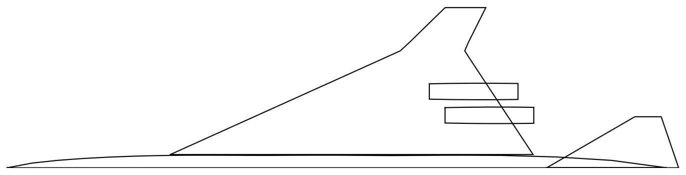

Figure 6. Optimum HSCT planform for 28 d.v. configuration.

The resulting optimum HSCT design configuration is plotted in Figure 6. The optimum design had a TOGW of 753,900 lbs and was similar to designs found in an earlier study [1] using a local optimizer. While more computational time was spent finding essentially the same design using the global optimizer as the local optimizer, an assurance was gained that the optimum found was the global optimum.

The parallel efficiencies for the runs are plotted in Figure 7. Efficiency is calculated relative to a serial implementation of DIRECT. With static load balancing, the efficiency starts high (97%) for 4 processors and then linearly decreases to 83% with all 64 processors. The master-slave organization of dynamic load balancing starts with a low efficiency, and then the efficiency gradually increases to be the highest of the load balancing schemes for 64 processors. The initial low values of efficiency are because, even though four processors are used, only the three slave processors are evaluating tasks. As the number of processors increases, the increased number of slave processors minimizes this effect. The fully distributed version with dynamic load balancing performs the best up to 32 processors and then the efficiency falls to 73% when using 64 processors. This is attributable to both the short average time per task and the relatively small amount of total work assigned to each of the 64 processors. The distributed version with threads performs the worst of all the methods, rapidly decreasing in efficiency as the number of processors used is increased. This behaviour was not observed for pthreads on the Intel Paragon reported in [9], and thus is more likely a reflection of the SGI pthreads implementation than of an inherent characteristic of pthreads.

To provide insight into why the distributed versions of the code were performing poorly for a large number of processors, a plot of the individual processor load for a complete optimization was made (Figure 8) for the 64 processor case. From this plot it is clear

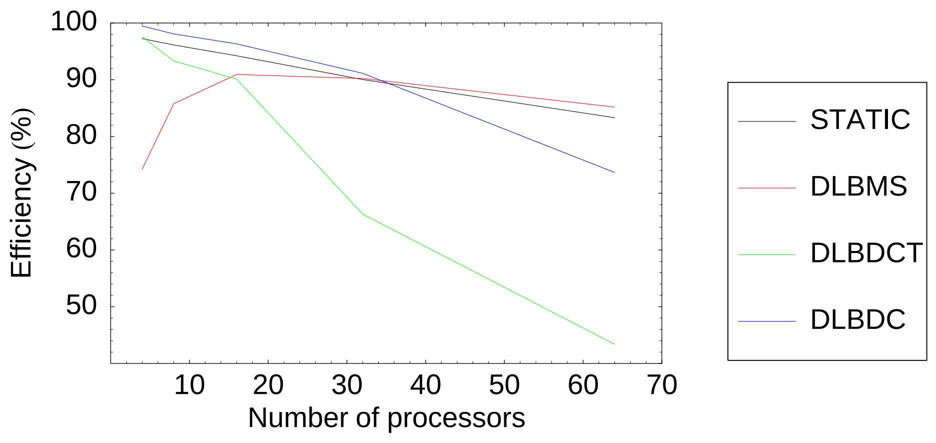

Figure 7. Parallel efficiencies for 4, 8, 16, 32, and 64 processor cases.

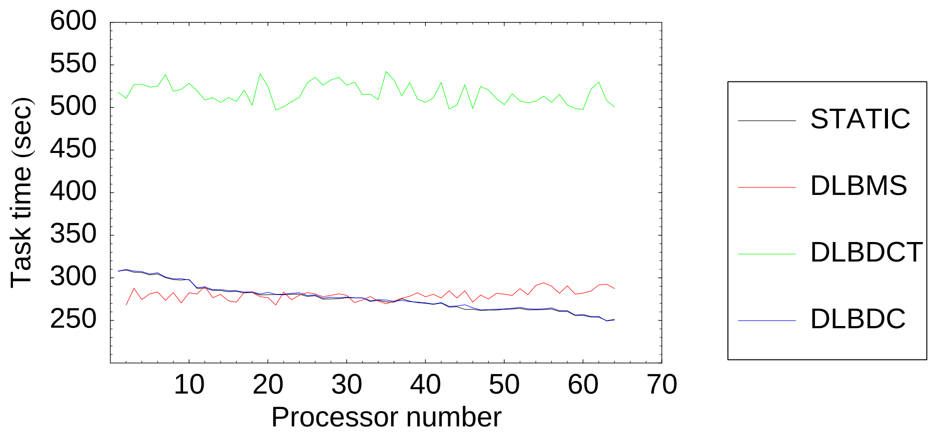

Figure 8. Individual processor load, 64 processor case.

that the master-slave organization (DLBMS) does the best job of load balancing, the curve being nearly horizontal. The load distribution for the distributed version without threads (DLBDC) falls directly on top of the curve for the static load balancing case (STATIC). This is due to the variation in function evaluation times being small enough that no tasks get transferred between processors, so DLBDC effectively becomes static load balancing. This effect does not appear when the number of processors is small because each processor has a larger set and with a large set the absolute differences in evaluation times are magnified (though relative differences may shrink) enough to where dynamic load balancing does take place. Since the distributed version DLBDC degrades to static load balancing STATIC for the 64 processor case and there is also computational overhead associated with the message passing, the overall efficiency is reduced. The time spent evaluating tasks for the threaded code DLBDCT is almost double that of all other methods. It was found that having the communicator thread continuously running sufficiently impeded the performance of each processor on the Origin to cause a noticeable rise in function evaluation times.

6 Conclusions

A Lipschitzian global optimization algorithm and a variety of parallel load balancing strategies were successfully integrated into a design space exploration method applied to a meaningful, complex aircraft design problem. The globally optimum HSCT design was found in slightly more than 10,000 function evaluations using an optimization algorithm that was able to thoroughly explore the design space. The load balancing methods implemented ranged from simple static load balancing to fully distributed dynamic load balancing via threads. It was observed that the static load balancing method was the most efficient for a large number of processors, due to the lack of variation in function evaluation times for the test problem. When the variation in function evaluation times is significant, as is the case for some other aircraft design problems [9], or as here when using a small number processors, the fully distributed dynamic load balancing method is most efficient. The use of pthreads greatly facilitates programming, but the execution efficiency of pthreads varies greatly between system implementations—from nearly invisible on the Intel Paragon to a factor of two slower on the SGI Origin.

Acknowledgements

Support for this research effort was provided through the NASA Ames Research Center grant NAG-2-1180.

References

[1] C.A. Baker, B. Grossman, R.T. Haftka, W.H. Mason, and L.T. Watson, "HSCT configuration design space exploration using aerodynamic response surface approximations," in *Proceedings of 7th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Saint Louis, MO, pages 769–777, September 1998.

[2] J. Bertin and M. Smith, *Aerodynamics for Engineers*, pages 261–282, Prentice Hall, 2nd edition, 1989.

[3] H. Carlson, R. Mack, and R. Barger, "Estimation of attainable leading edge thrust for wings at subsonic and supersonic speeds," Technical Report NASA TP-1500, 1979.

[4] H. Carlson and D. Miller, "Numerical methods for the design and analysis of wings at supersonic speeds," Technical Report NASA TN D-7713, 1974.

[5] R. Harris Jr, "An analysis and correlation of aircraft wave drag," Technical Report NASA TM X-947, 1964.

[6] M.G. Hutchison, W.H. Mason, R.T. Haftka, and B. Grossman, "Aerodynamic optimization of an HSCT configuration using variable-complexity modeling," AIAA 31st Aerospace Sciences Meeting and Exhibit, Reno, NV, AIAA Paper 93-0101, January 1993.

[7] M.G. Hutchison, E.R. Unger, W.H. Mason, B. Grossman, and R.T. Haftka, "Variable-complexity aerodynamic optimization of a high-speed civil transport wing," *Journal of Aircraft*, volume 31(1), pages 110–116, 1994.

[8] D.R. Jones, C.D. Perttunen, and B.E. Stuckman, "Lipschitzian optimization without the Lipschitz constant," *Journal of Optimization Theory and Application*, volume 79(1), pages 157–181, October 1993.

[9] D.T. Krasteva, C. Baker, L.T. Watson, B. Grossman, W.H. Mason, and R.T. Haftka, "Distributed control parallelism for multidisciplinary design of a high speed civil transport," in *Proceedings of 7th Symposium on the Frontiers of Massively Parallel Computation*, IEEE Computer Society, Los Alamitos, CA, pages 166–173, April 1999.

[10] P. MacMillin, O. Golovidov, W. Mason, B. Grossman, and R. Haftka, "Trim, control, and performance effects in variable-complexity high-speed civil transport design," Technical Report MAD 96-07-01, Virginia Polytechnic Institute and State University, Blacksburg, VA, July 1996.

[11] P.E. MacMillin, O.B. Golovidov, W.H. Mason, B. Grossman, and R.T. Haftka, "An MDO investigation of the impact of practical constraints on an HSCT optimization," AIAA 35th Aerospace Sciences Meeting and Exhibit, Reno, NV, AIAA Paper 97-0098, January 1997.

[12] L.A. McCullers, "Aircraft configuration optimization including optimized flight profiles," in *Proceedings of a Symposium on Recent Experiences in Multidisciplinary Analysis and Optimization*, NASA CP-2327, pages 395–412, April 1984.

[13] M. Snir, S. Otto, S. Huss-Lederman, D.W. Walker, and J. Dongarra, *MPI: The Complete Reference*, MIT Press, Cambridge, MA, 1996.