A FULLY DISTRIBUTED PARALLEL GLOBAL SEARCH ALGORITHM

Layne T. Watson
Multidisciplinary Analysis and Design Center for Advanced Vehicles
Virginia Polytechnic Institute and State University, Virginia, USA
Chuck A. Baker
Engineous Software, Inc., North Carolina, USA

Keywords *Direct search, Distributed control, Global optimization, Parallel algorithm*

Abstract *The n-dimensional direct search algorithm DIRECT of Jones, Perttunen, and Stuckman has attracted recent attention from the multidisciplinary design optimization community. Since DIRECT only requires function values (or ranking) and balances global exploration with local refinement better than n-dimensional bisection, it is well suited to the noisy function values typical of realistic simulations. While not efficient for high accuracy optimization, DIRECT is appropriate for the sort of global design space exploration done in large scale engineering design. Direct and pattern search schemes have the potential to exploit massive parallelism, but efficient use of massively parallel machines is nontrivial to achieve. This paper presents a fully distributed control version of DIRECT that is designed for massively parallel (distributed memory) architectures. Parallel results are presented for a multidisciplinary design optimization problem—configuration design of a high speed civil transport.*

1. Introduction

There has been a renaissance in direct search algorithms for optimization, as nicely summarized in the recent state-of-the-art assessment by Lewis et al. (2000). Multidisciplinary design optimization (MDO), or any engineering design enterprise based on large scale high fidelity simulation codes, encounters nonsmooth functions and the need to combine global design space exploration (using low fidelity analyses) with local refinement (using high or medium fidelity) to accurately optimize the final design. Just as a range of physical models should be used for the design process (the "variable complexity" concept), so should a variety of optimization techniques be employed. The suggestion here is that direct search algorithms, coupled with massively parallel computation, be used for global design space exploration to identify promising regions, which would then be investigated using derivative based optimization or response surface methods.

The n-dimensional direct search algorithm DIRECT of Jones, Perttunen, and Stuckman (1993) has attracted recent attention from the multidisciplinary design optimization community. Since DIRECT only requires function values (or ranking) and balances global exploration with local refinement better than n-dimensional bisection, it is well suited to the noisy function values typical of realistic simulation codes. While not efficient for high accuracy optimization, DIRECT is appropriate for the sort of global design space exploration done in large scale engineering design. Direct and pattern search schemes have the potential to exploit massive parallelism, but efficient use of massively parallel machines is nontrivial to achieve.

While it is not difficult to achieve small scale parallelism with shared memory architectures or a master-slave programming paradigm, large scale (speedups > 1000) parallelism is another matter entirely. Even with several hundred processors (as the results presented later here show), memory contention in a shared memory machine dominates the computation, and such architectures clearly will not scale. Similarly, as the number of tasks grows, eventually the master-slave model results in a communication bottleneck at the master node, and efficiency drops sharply. The purpose of this paper is to describe, in some detail, a parallel version of DIRECT that will scale to thousands

of processors (in a distributed memory machine). This is accomplished by using fully distributed control and message passing.

Section 2 describes the serial DIRECT algorithm as proposed by Jones et al. Sections 3 and 4 provide background on the programming techniques used for fully distributed control. For completeness and for comparison, Section 5 gives pseudo code for a parallel master-slave version of DIRECT. Section 6 then follows with the detailed description of the parallel distributed control version of DIRECT. A more aggressive parallel version is suggested later when discussing performance results. Section 7 very briefly describes a high speed civil transport (HSCT) application, for which parallel performance results are presented in Section 8. Section 9 summarizes the results and concludes.

2. Serial DIRECT algorithm

Jones et al. (1993) describe their search algorithm DIRECT as a Lipschitzian unconstrained optimization algorithm that (effectively) uses all possible values of the Lipschitz constant. By using different values of the constant, which can be viewed as an upper limit on the variation of the function, 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 rectangle or box. All boxes are identified by their center point c_i and their function value $f(c_i)$ 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 directions 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. The algorithm (Jones et al. (1993)) is as follows:

1. Normalize the search space to be the unit hypercube. Let c_1 be the center point of this hypercube and evaluate $f(c_1)$.
2. Identify the set S of potentially optimal rectangles (those rectangles defining the bottom of the convex hull of a scatter plot of rectangle diameter versus $f(c_i)$ for all rectangle centers c_i) as in Figure 1.
3. For all rectangles $j \in S$:
   - 3a. Identify the set I of dimensions with the maximum side length. Let δ equal one-third of this maximum side length.
   - 3b. Sample the function at the points $c \pm \delta e_i$ for all $i \in I$, where c is the center of the rectangle and e_i is the ith unit vector.
   - 3c. Divide the rectangle containing c into thirds along the dimensions in I, starting with the dimension with the lowest value of $f(c \pm \delta e_i)$ and continuing to the dimension with the highest $f(c \pm \delta e_i)$.

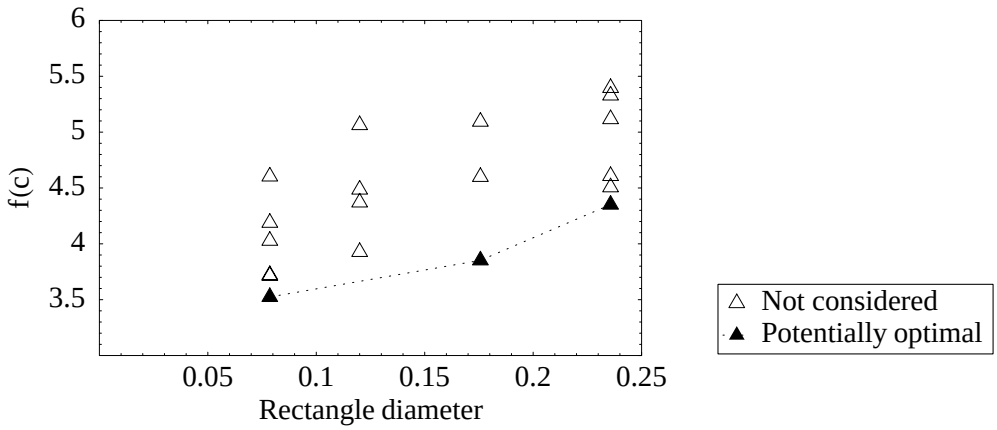

Figure 1. Rectangles selected for further subdivision by DIRECT.

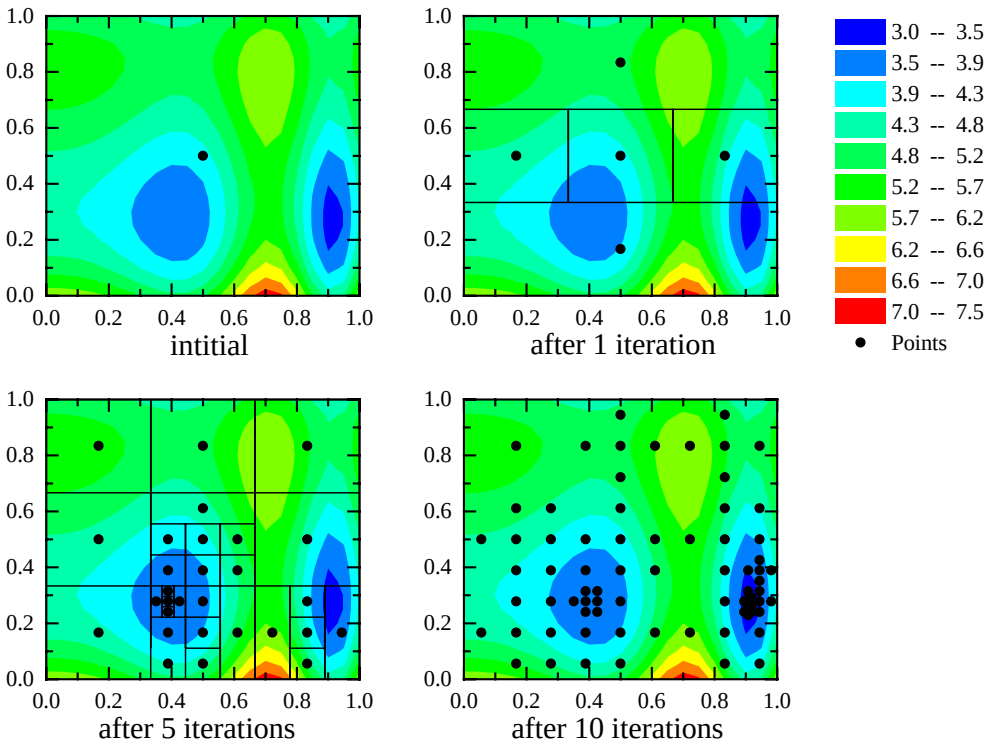

Figure 2. Illustration of DIRECT behavior with multiple local optima.

4. Repeat 2.–3. until stopping criterion is met.

DIRECT's behavior on a simple 2–D test function is shown in Figure 2. The test function has local minima at (0.4, 1.0), (0.9, 0.3), 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 work, the algorithm was run for a fixed number of iterations. Constraints can be accounted for through the use of penalty functions, but undefined values $f(c)$ (e.g., when a simulation fails due to data outside the intended operating range) require more subtle handling.

3. 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) (Snir et al. (1996)), 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. To investigate this effect, a version of the master-slave implementation was also used that distributes the tasks in bins of 10.

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 root processor, P_0, 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 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 (Krasteva et al. (1999)), 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 is detected. If a work request is received by a processor, half of its remaining tasks are transferred to the requesting processor. Also, a characteristic of random polling described in Tel (1994) is exploited, where it is beneficial to use random polling if the number of tasks to load balanced is greater than $N_p \log N_p$, where N_p is the total number of processors . If the number of potentially optimal boxes meets this criteria, then it is the potentially optimal boxes that are deemed tasks.

If this number isn't large enough, as is usually the case, then the newly sampled points inside the potentially optimal boxes are load balanced.

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—bin size 1 (DLBMS01), dynamic load balancing with the master-slave paradigm—bin size 10 (DLBMS10), dynamic load balancing with fully distributed control (DLBDC), dynamic load balancing with fully distributed control using pthreads (DLBDCT).

4. Termination detection

The termination detection scheme used for DLBDC and DLBDCT is the standard token wave algorithm used in Krasteva et al. (1999). Suppose there are P processors. Each processor keeps track of its state in a local flag $idle$. Initially, the flag $idle$ is set to false if a processor has work or true otherwise. If at any time a processor receives work, the $idle$ flag is set to false. A token is passed around, in ring fashion, to all processors. If a processor with $idle = true$ receives the token, the token is less than P, and there are no pending requests for incoming work, the token value is incremented and sent to the next processor in the ring; if the token received is equal to P, then that processor terminates, and broadcasts a termination message to all other processors. If a processor with $idle = false$ receives the token, the token is set to zero. When that processor finishes its work, it passes the (zero) token along and sets $idle = true$. After all the tasks on all the processors have been completed, the token makes two complete circuits of the ring of processors, terminating at the end of the second circuit.

5. Parallel master-slave pseudo code DLBMS01

Below is pseudo code for a parallel implementation of DIRECT incorporating the hierarchical, centralized control, master-slave paradigm DLBMS01. This is a precise description of the master-slave parallel version of DIRECT described in Section 3.

```
iteration := 1
while iteration ≤ maximum iteration
  if P_0 then
    find potentially optimal point set, local C_opt, from previously evaluated box
      center points, C_eval
    sample around all C_opt to create C_new
    total tasks N_tasks := number of points in C_new
    sent task counter i_sent := 0
    received task counter i_recv := 0
    slave processor counter P_slave := 1
    while P_slave ≤ number of slave processors
      if P_slave ≤ N_tasks then
        i_sent := i_sent + 1
        send C_new(i_sent) to P_slave
      else
        send dummy point with termination tag to P_slave
      end if
      P_slave := P_slave + 1
    end while
    while i_recv < N_tasks
      receive i_task, function value at i_task, and processor id,
        P_slave, from any slave processor
      set C_new(i_task) := function value at i_task
      i_recv := i_recv + 1
      if i_sent < N_tasks then
        i_sent := i_sent + 1
        send C_new(i_sent) to P_slave
      else
        send dummy point with termination tag to P_slave
      end if
    end while
    set new box side lengths for C_new and its parent C_opt points
    append all C_new to C_eval
  else
    while termination tag not received
      receive C_new point from P_0
      if termination tag not received then
        evaluate function at point
        send point id, function at point, and processor id to P_0
      end if
    end while
  end if
  iteration := iteration + 1
end while
```

6. Parallel distributed control pseudo code DLBDC

Below is pseudo code for a parallel implementation of DIRECT using N_p processors, incorporating fully distributed control (DLBDC). This is a precise description of the parallel distributed control version of DIRECT described in Sections 3 and 4.

```
iteration := 1
while iteration ≤ maximum iteration
    find potentially optimal point set, local C_opt, from previously evaluated box
        center points, C_eval
    if P_0 then
        gather C_opt from all processors
        find global C_opt from local C_opt sets
        broadcast global C_opt set to all processors
    end if
    remove points in local C_opt not in global C_opt
    if number of points in global C_opt ≤ N_p log N_p, then
        total tasks N_tasks := number of points in local C_opt
        task counter i_task := 1
        while termination not detected
            if i_task ≤ N_tasks then
                sample around all C_opt(i_task) to create C_new(i_task)
                evaluate function at all C_new(i_task)
                i_task := i_task + 1
            else
                if outgoing work request is not pending, then
                    generate random processor number, P_rand
                    send work request to P_rand
                end if
            end if
            process message of each type (incoming work request, outgoing work
                request reply, token pass, etc.) received;
            if outgoing work request reply received then increment N_tasks
                by number of tasks received;
            if work request received then
                if N_tasks − i_task > 1 then
                    send ⌊(N_tasks − i_task)/2⌋ tasks to requesting processor;
                    decrement N_tasks by ⌊(N_tasks − i_task)/2⌋ tasks;
                else
                    send 0 tasks to requesting processor;
                end if
            end if
        end while
    else
        sample around all C_opt to create C_new
        total tasks N_tasks := number of points in C_new
        task counter i_task := 1
```

```
        while termination not detected
            if i_task ≤ N_tasks then
                evaluate function at C_new(i_task)
                i_task := i_task + 1
            else
                if outgoing work request is not pending, then
                    generate random processor number, P_rand
                    send work request to P_rand
                end if
            end if
            process message of each type (incoming work request, outgoing work
                request reply, token pass, etc.) received;
            if outgoing work request reply received then increment N_tasks
                by number of tasks received;
            if work request received then
                if N_tasks − i_task > 1 then
                    send ⌊(N_tasks − i_task)/2⌋ tasks to requesting processor;
                    decrement N_tasks by ⌊(N_tasks − i_task)/2⌋ tasks;
                else
                    send 0 tasks to requesting processor;
                end if
            end if
        end while
    end if
    if P_0 then
            gather C_new from all processors
        sort C_new points by parent processor rank
        scatter each C_new point to its parent processor
    end if
    set new box side lengths for C_new and its parent C_opt points
    append all C_new to C_eval
    iteration := iteration + 1
end while
```

7. HSCT configuration design application

The application chosen to illustrate the parallel versions of DIRECT is the optimization of a HSCT configuration (MacMillin et al. (1996), MacMillin et al. (1997)) 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. 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 dictates drag and thus the required fuel weight. This HSCT design problem is described in detail elsewhere (Baker et al. (1998), Golovidov (1997), Hutchison et al. (1993), Hutchison et al. (1994), MacMillin et al. (1996), MacMillin et al. (1997)), and thus only a few pertinent details will be repeated here.

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

Table 1. HSCT configuration design variables and limits.

Index	Description	Lower	Upper
1	Wing root chord (ft)	130	200
2	Leading edge (LE) break point, streamwise (ft)	50	160
3	LE break point, spanwise (% of semispan)	40	95
4	Wing break chord (ft)	7	50
5	LE wing tip, streamwise (ft)	50	180
6	Wing tip chord (ft)	7	30
7	Wing semispan (ft)	50	90
8	Chordwise location of max. thickness (% of chord)	20	60
9	LE radius parameter	1.5	4.0
10	Airfoil t/c at wing root	0.0150	0.0350
11	Airfoil t/c at wing break	0.0150	0.0300
12	Airfoil t/c at wing tip	0.0150	0.0250
13	Fuselage restraint 1, streamwise (ft)	2.00	8.00
14	Fuselage restraint 1, radius (ft)	0.4	0.8
15	Fuselage restraint 2, streamwise (ft)	12	18
16	Fuselage restraint 2, radius (ft)	2.0	4.0
17	Fuselage restraint 3, streamwise (ft)	100	140
18	Fuselage restraint 3, radius (ft)	4.0	7.0
19	Fuselage restraint 4, streamwise (ft)	150	250
20	Fuselage restraint 4, radius (ft)	4.0	7.0
21	Nacelle 1 location (7 ft + % of spanwise break)	0	80
22	Nacelle 2 location (DV 21 + % of DV 3 - DV 21)	0	100
23	Flight fuel (lb)	200,000	600,000
24	Starting cruise altitude (ft)	50,000	70,000
25	Cruise climb rate (ft/min)	25.0	45.0
26	Vertical tail area (ft^2)	400	900
27	Horizontal tail area (ft^2)	600	1,200
28	Thrust per engine (lb)	3,000	7,000

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.

The n-dimensional design *box* (normalized by DIRECT) uses the variable limits also given in Table 1. In order to ensure that a thorough design space exploration was being conducted, the bounds were chosen to include as wide 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.

The methods used to calculate the drag components used in the drag calculation and their corresponding ranges are described in Hutchison et al. (1993) and Hutchison et al. (1994). The

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 ≥ 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 ≥ 5500 nautical miles
35	C_L at landing speed ≤ 1
36–53	Section C_L at landing ≤ 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 ≤ 11000 ft
67–68	Mission segments: thrust available $\geq$ thrust required

aerodynamics calculations are based on the Mach box method (Carlson et al. (1974), Carlson et al. (1979)), and the Harris (1964) wave drag code. A simple strip boundary layer friction estimate is implemented as in Hutchison et al. (1994). A vortex lattice method with vortex lift and ground effects included (Bertin et al. (1989)) is used to calculate landing angle of attack. Structural weights are calculated by the FLOPS (McCullers (1984)) weight equations. Each of these analysis methods uses iterative algorithms or discretization methods that can cause differences in the time needed to evaluate different HSCT designs, hence the need for dynamic load balancing.

Figure 3 shows a planar slice in the 28-dimensional design space, illustrating that the feasible set is nonconvex and possibly even multiply connected. The need for a global exploration algorithm like DIRECT is clear.

8. Parallel performance

The parallel runs were conducted on an SGI Origin 2000 with a total of 256 CPUs. Runs were made on 4, 8, 16, 32, and 64 processors for each of the five load balancing methods. The DIRECT optimizer was terminated after 37 iterations, performing 10,077 function evaluations. The parallel efficiencies for the runs are plotted in Figure 4. Efficiency is calculated relative to a serial implementation of DIRECT. With static load balancing, the efficiency starts high (0.97) for 4 processors and then linearly decreases to 0.83 with all 64 processors. The master-slave organization

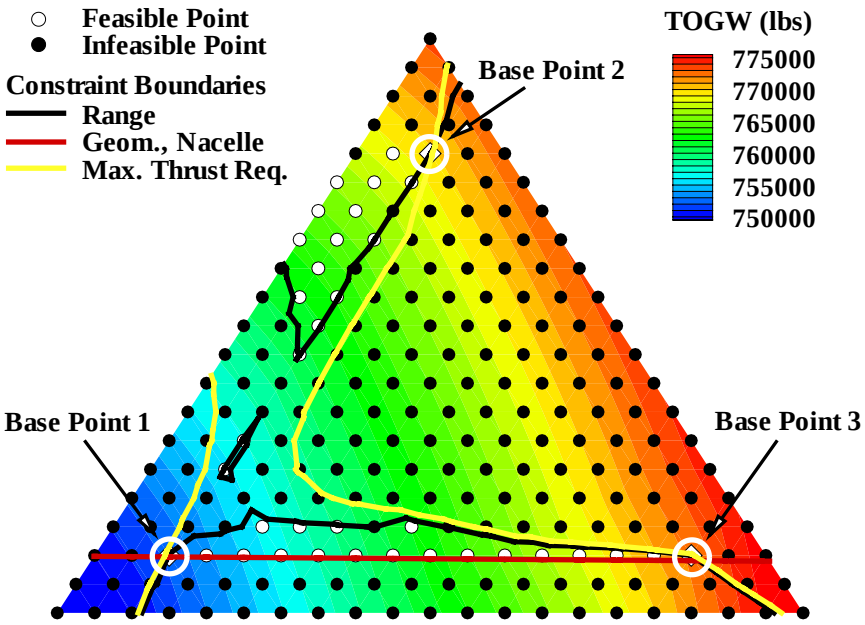

Figure 3. Design space visualization.

(DLBMS01) 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 master-slave organization with a bin size of 10 (DLBMS10) initially has a low efficiency like DLBMS01 then it peaks at 0.80 for 8 processors. From then as the number of processors used increases, the efficiency plateaus at 0.57. The fully distributed version with dynamic load balancing performs the best up to 32 processors and then the efficiency drops to 0.84 when using 64 processors, slightly below that of DLBMS01 and slightly above that of STATIC. 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. Also, a peculiarity was observed in that DLBDC either ran at an efficiency of 0.84 or 0.78 (shown on plot). 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 Krasteva et al. (1999), and thus is more likely a reflection of the SGI pthreads implementation than of an inherent characteristic of pthreads.

A detailed discussion of these parallel performance results, relative to the SGI Origin hardware and process assignment to memory banks and CPUs, can be found in Baker (2000). In essence, the Origin results for DLBDC are not indicative of what would occur (on a distributed memory machine) as the number of tasks, the time per task, and the number of processors are all increased.

To observe the effect of larger sets of tasks for a large number of processors, a more aggressive version of the DIRECT algorithm is implemented. For the aggressive DIRECT, the idea of using the Lipschitz constants is discarded and the box with the smallest objective function for each box size existing is deemed potentially optimal and subsequently subdivided. Consequently, for the example shown in Figure 1 there will be a total of four potentially optimal boxes like in Figure 5,

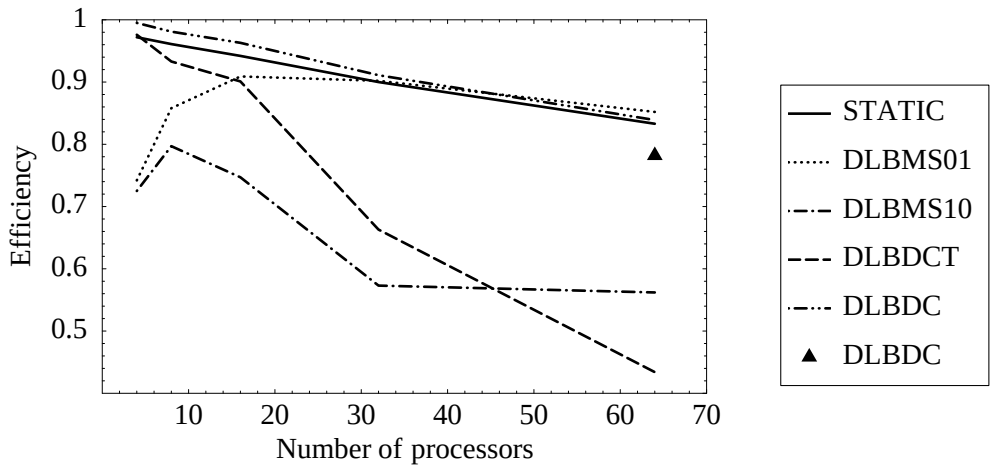

Figure 4. Parallel efficiencies for 4, 8, 16, 32, and 64 processor cases.

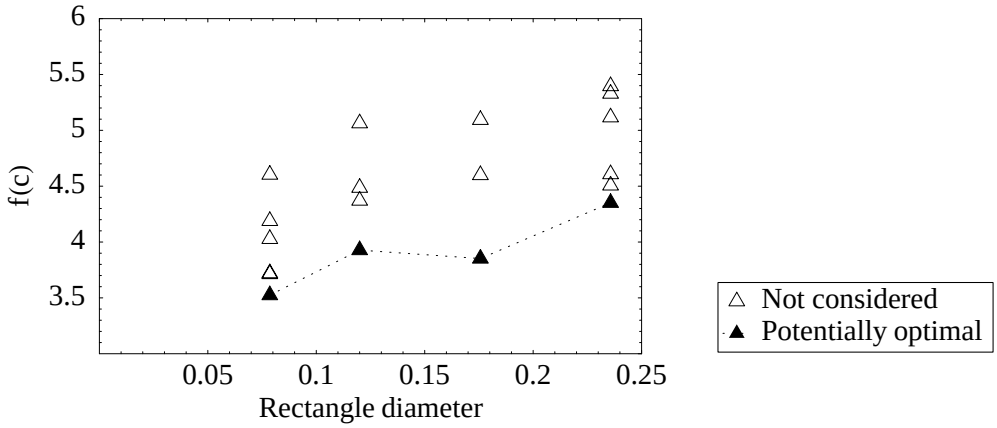

Figure 5. Rectangles selected for further subdivision by aggressive DIRECT.

instead of the three for the standard DIRECT algorithm. This change in the algorithm typically results in a much larger set of new tasks to be evaluated and load balanced at each iteration.

Aside from generating a large set of data at each iteration, this method also has the benefit of being able to find the global optimum in fewer iterations. A box containing the global optimum can be neglected using the standard DIRECT algorithm due to the way that the design variable limits are chosen. A box containing the global optimum may be potentially optimal with one set of variable limits and look poor for another. With the aggressive version, since a box of each size is subdivided, the effect of the variable limits is less substantial. If the variable limits chosen make a box look poor, the box is still more likely to be subdivided and the global optimum found.

The parallel runs were conducted on the same SGI Origin 2000 as the standard DIRECT. Runs were made on 8, 16, 32, 64, and 128 processors for each of the five load balancing methods. Due to the large number of points generated, the DIRECT optimizer was terminated after 20 iterations and performing 48,577 function evaluations. Compared to the standard DIRECT, the variation in evaluation time per task is increased as well as the number of tasks with aggressive DIRECT. The aggressive version was also able to find a better optimum HSCT design in fewer iterations than standard DIRECT, although of course the total number of evaluations and aggregate CPU time

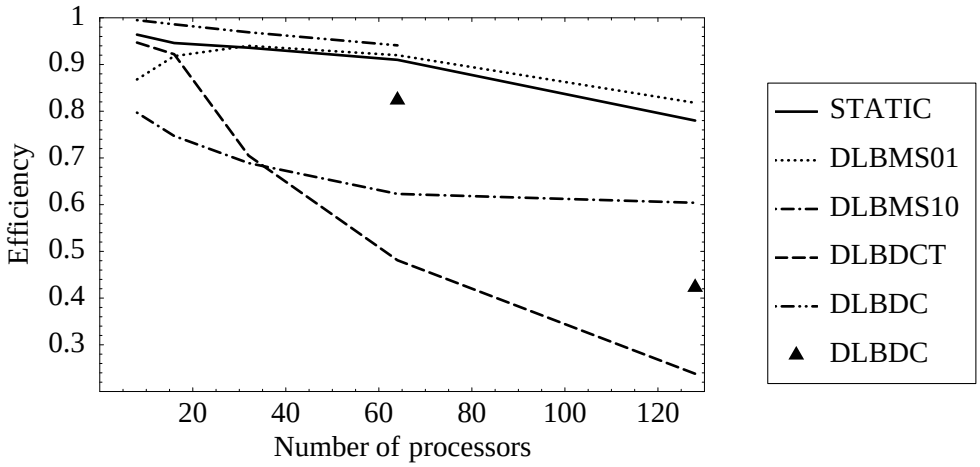

Figure 6. Parallel efficiencies for 8, 16, 32, 64, and 128 processor cases.

are more (the aggressive case used 86,374 seconds of serial CPU time versus 17,642 seconds for the standard DIRECT).

The parallel efficiencies for the runs using the aggressive DIRECT are plotted in Figure 6. All the load balancing methods implemented exhibit similar trends as when used with the standard DIRECT except that their efficiencies have been slightly improved. Due to the increase of variation in evaluation time, DLBDC is now the most efficient method to 64 processors, where its efficiency is 0.94. The anomaly for DLBDC (attributable to SGI hardware and scheduling algorithms) was again observed, and no good run for 128 processors ever occurred (cf. the solid triangles in Figure 6).

9. Conclusions

A variety of parallel load balancing strategies were successfully integrated into a global design space exploration method and applied to a meaningful, complex aircraft design problem. The load balancing methods implemented ranged from simple static load balancing to fully distributed dynamic load balancing via pthreads. It was observed that the master-slave load balancing method was the most efficient for a large number of processors, when the variation in function evaluation times was small (because the master then did little redistribution work). When the variation in function evaluation times is significant, as is the case for the aggressive DIRECT algorithm or inherently in other aircraft design problems (Krasteva et al. (1999)), or as here when using a small number of processors, the fully distributed dynamic load balancing method is most efficient. The implication is that for large scale realistic MDO problems, compared to static distribution or dynamic load balancing via a master-slave paradigm, the fully distributed control paradigm for dynamic load balancing will scale the best to massively parallel (distributed memory) machines. One final practical observation is that 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.

References

C. A. Baker, B. Grossman, R. T. Haftka, W. H. Mason, and L. T. Watson(1998), "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, pp. 769-777.

C. A. Baker(2000), "Parallel global aircraft configuration design space exploration", Technical Report MAD 2000-06-28, Virginia Polytechnic Institute and State University, Blacksburg, VA.

J. Bertin and M. Smith(1989), *Aerodynamics for Engineers*, Prentice Hall.

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

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

O. Golovidov(1997), "Variable-Complexity Response Surface Approximations for Aerodynamic Parameters in HSCT Optimization", Master's thesis, Virginia Polytechnic Institute and State University, VA.

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

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

M. G. Hutchison, E. R. Unger, W. H. Mason, B. Grossman, and R. T. Haftka(1994), "Variable-complexity aerodynamic optimization of a high-speed civil transport wing", *Journal of Aircraft*, Vol. 31, No. 1, pp. 110–116.

D. R. Jones, C. D. Perttunen, and B. E. Stuckman(1993), "Lipschitzian optimization without the Lipschitz constant", *Journal of Optimization Theory and Application*, Vol. 79, No. 1, pp. 157–181.

D. T. Krasteva, C. Baker, L. T. Watson, B. Grossman, W. H. Mason, and R. T. Haftka(1999), "Distributed control parallelism in multidisciplinary aircraft design", *Concurrency: Practice and Experience*, Vol. 11, pp. 435–459.

R. L. Lewis, V. Torczon, and M. W. Trosset, "Direct search methods: then and now", *Journal of Computational and Applied Mathematics*, to appear.

P. MacMillin, O. Golovidov, W. Mason, B. Grossman, and R. Haftka(1996), "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.

P. E. MacMillin, O. B. Golovidov, W. H. Mason, B. Grossman, and R. T. Haftka(1997), "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.

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

M. Snir, S. Otto, S. Huss-Lederman, D. W. Walker, and J. Dongarra(1996), *MPI: The Complete Reference*, MIT Press.

G. Tel(1994), *Introduction to Distributed Algorithms*, Cambridge University Press.