INTERNATIONAL JOURNAL FOR NUMERICAL METHODS IN ENGINEERING
Int. J. Numer. Meth. Engng 2004; **61**:2296–2315
Published online 18 October 2004 in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/nme.1149

Parallel global optimization with the particle swarm algorithm

J. F. Schutte[1], J. A. Reinbolt[2], B. J. Fregly[1,2,*,†,‡], R. T. Haftka[1,‡] and A. D. George[3,‡]

[1]*Department of Mechanical and Aerospace Engineering, University of Florida, Gainesville, FL, U.S.A.*
[2]*Department of Biomedical Engineering, University of Florida, Gainesville, FL, U.S.A.*
[3]*Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL, U.S.A.*

SUMMARY

Present day engineering optimization problems often impose large computational demands, resulting in long solution times even on a modern high-end processor. To obtain enhanced computational throughput and global search capability, we detail the coarse-grained parallelization of an increasingly popular global search method, the particle swarm optimization (PSO) algorithm. Parallel PSO performance was evaluated using two categories of optimization problems possessing multiple local minima—large-scale analytical test problems with computationally cheap function evaluations and medium-scale biomechanical system identification problems with computationally expensive function evaluations. For load-balanced analytical test problems formulated using 128 design variables, speedup was close to ideal and parallel efficiency above 95% for up to 32 nodes on a Beowulf cluster. In contrast, for load-imbalanced biomechanical system identification problems with 12 design variables, speedup plateaued and parallel efficiency decreased almost linearly with increasing number of nodes. The primary factor affecting parallel performance was the synchronization requirement of the parallel algorithm, which dictated that each iteration must wait for completion of the slowest fitness evaluation. When the analytical problems were solved using a fixed number of swarm iterations, a single population of 128 particles produced a better convergence rate than did multiple independent runs performed using sub-populations (8 runs with 16 particles, 4 runs with 32 particles, or 2 runs with 64 particles). These results suggest that (1) parallel PSO exhibits excellent parallel performance under load-balanced conditions, (2) an asynchronous implementation would be valuable for real-life problems subject to load imbalance, and (3) larger population sizes should be considered when multiple processors are available. Copyright © 2004 John Wiley & Sons, Ltd.

KEY WORDS: particle swarm; parallel global optimization; cluster computing

*Correspondence to: B. J. Fregly, Department of Mechanical and Aerospace Engineering, 231 MAEA Building, P.O. Box 116250, University of Florida, Gainesville, FL 32611-6250, U.S.A.
†E-mail: fregly@ufl.edu
‡Ph.D.

Contract/grant sponsor: NIH National Library of Medicine; contract/grant number: R03 LM07332
Contract/grant sponsor: Whitaker Foundation
Contract/grant sponsor: AFOSR; contract/grant number: F49620-09-1-0070

Received 10 July 2003
Revised 23 February 2004
Accepted 7 June 2004

INTRODUCTION

Numerical optimization has been widely used in engineering to solve a variety of NP-complete problems in areas such as structural optimization, neural network training, control system analysis and design, and layout and scheduling problems. In these and other engineering disciplines, two major obstacles limiting the solution efficiency are frequently encountered. First, even medium-scale problems can be computationally demanding due to costly fitness evaluations. Second, engineering optimization problems are often plagued by multiple local optima, requiring the use of global search methods such as population-based algorithms to deliver reliable results.

Fortunately, recent advances in microprocessor and network technology have led to increased availability of low cost computational power through clusters of low to medium performance computers. To take advantage of these advances, communication layers such as MPI [1, 2] and PVM [3] have been used to develop parallel optimization algorithms, the most popular being gradient-based, genetic (GA), and simulated annealing (SA) algorithms [4–6]. In biomechanical optimizations of human movement, for example, parallelization has allowed problems requiring days or weeks of computation on a single-processor computer to be solved in a matter of hours on a multi-processor machine [4].

The particle swarm optimization (PSO) algorithm is a recent addition to the list of global search methods [7]. This derivative-free method is particularly suited to continuous variable problems and has received increasing attention in the optimization community. It has been successfully applied to large-scale problems [8–10] in several engineering disciplines and, being a population-based approach, is readily parallelizable. It has few algorithm parameters, and generic settings for these parameters work well on most problems [11, 12].

In this study, we present a parallel PSO algorithm for application to computationally demanding optimization problems. The algorithm's enhanced throughput due to parallelization and improved convergence due to increased population size are evaluated using large-scale analytical test problems and medium-scale biomechanical system identification problems. Both types of problems possess multiple local minima. The analytical test problems utilize 128 design variables to create a tortuous design space but with computationally cheap fitness evaluations. In contrast, the biomechanical system identification problems utilize only 12 design variables but each fitness evaluation is much more costly computationally. These two categories of problems provide a range of load balance conditions for evaluating the parallel formulation.

SERIAL PARTICLE SWARM ALGORITHM

Particle swarm optimization was introduced in 1995 by Kennedy and Eberhart [13]. Although several modifications to the original swarm algorithm have been made to improve performance [14–18] and adapt it to specific types of problems [9, 19, 20], a parallel version has not been previously implemented.

The following is a brief introduction to the operation of the PSO algorithm. Consider a swarm of p particles, with each particle's position representing a possible solution point in the design problem space D. For each particle i, Kennedy and Eberhart proposed that its

position $\mathbf{x}^i$ be updated in the following manner:

$$\mathbf{x}_{k+1}^i = \mathbf{x}_k^i + \mathbf{v}_{k+1}^i \tag{1}$$

with a pseudo-velocity $\mathbf{v}_{k+1}^i$ calculated as follows:

$$\mathbf{v}_{k+1}^i = w_k \mathbf{v}_k^i + c_1 r_1 (\mathbf{p}_k^i - \mathbf{x}_k^i) + c_2 r_2 (\mathbf{p}_k^g - \mathbf{x}_k^i) \tag{2}$$

Here, subscript k indicates a (unit) pseudo-time increment, $\mathbf{p}_k^i$ represents the best ever position of particle i at time k (the cognitive contribution to the pseudo-velocity vector $\mathbf{v}_{k+1}^i$), and $\mathbf{p}_k^g$ represents the global best position in the swarm at time k (social contribution). r_1 and r_2 represent uniform random numbers between 0 and 1. To allow the product $c_1 r_1$ or $c_2 r_2$ to have a mean of 1, Kennedy and Eberhart proposed that the cognitive and social scaling parameters c_1 and c_2 be selected such that $c_1 = c_2 = 2$. The result of using these proposed values is that the particles overshoot the target half the time, thereby maintaining separation within the group and allowing for a greater area to be searched than if no overshoot occurred.

A modification by Fourie and Groenwold [9] on the original PSO algorithm [15] allows transition to a more refined search as the optimization progresses. This operator reduces the maximum allowable velocity $v_k^{\max}$ and particle inertia w_k in a dynamic manner, as dictated by the dynamic reduction parameters κ, d, w_d. For the sake of brevity, further details of this operater are omitted, but a detailed description can be found in References [9, 11].

The serial PSO algorithm as it would typically be implemented on a single CPU computer is described below, where p is the total number of particles in the swarm. The best ever fitness value of a particle at design co-ordinates $\mathbf{p}_k^i$ is denoted by f_{best}^i and the best ever fitness value of the overall swarm at co-ordinates $\mathbf{p}_k^g$ by f_{best}^g. At time step $k = 0$, the particle velocities $\mathbf{v}_0^i$ are initialized to values within the limits $0 \leqslant \mathbf{v}_0 \leqslant \mathbf{v}_0^{\max}$. The vector $\mathbf{v}_0^{\max}$ is calculated as a fraction of the distance between the upper and lower bounds $\mathbf{v}_0^{\max} = \zeta(\mathbf{x}_{\text{UB}} - \mathbf{x}_{\text{LB}})$ [9], with $\zeta = 0.5$. With this background, the PSO algorithm flow can be described as follows:

1. Initialize
   (a) Set constants $k_{\max}$, c_1, c_2, κ, $\mathbf{v}_0^{\max}$, w_k d, and w_d
   (b) Initialize dynamic maximum velocity $v_k^{\max}$ and inertia w_k
   (c) Set counters $k = 0$, $t = 0$, $i = 1$. Set random number seed
   (d) Randomly initialize particle positions $\mathbf{x}_0^i \in \mathbf{D}$ in $\mathbb{R}^n$ for $i = 1, \ldots, p$
   (e) Randomly initialize particle velocities $0 \leqslant \mathbf{v}_0^i \leqslant \mathbf{v}_0^{\max}$ for $i = 1, \ldots, p$
   (f) Evaluate fitness values f_0^i using design space co-ordinates $\mathbf{x}_0^i$ for $i = 1, \ldots, p$
   (g) Set $f_{\text{best}}^i = f_0^i$, $\mathbf{p}^i = \mathbf{x}_0^i$ for $i = 1, \ldots, p$
   (h) Set f_{best}^g to best f_{best}^i and $\mathbf{g}_0$ to corresponding $\mathbf{x}_0^i$
2. Optimize
   (a) Update particle velocity vector $\mathbf{v}_{k+1}^i$ using Equation (2)
   (b) Update particle position vector $\mathbf{x}_{k+1}^i$ using Equation (1)
   (c) Update dynamic maximum velocity $v_k^{\max}$ and inertia w_k
   (d) Evaluate fitness value f_k^i using design space co-ordinates $\mathbf{x}_k^i$

(e) If $f_k^i \leqslant f_{\text{best}}^i$ then $f_{\text{best}}^i = f_k^i$, $\mathbf{p}^i = \mathbf{x}_k^i$
(f) If $f_k^i \leqslant f_{\text{best}}^g$ then $f_{\text{best}}^g = f_k^i$, $\mathbf{p}^g = \mathbf{x}_k^i$
(g) If f_{best}^g was improved in (e) then reset $t = 0$. Else increment t
(h) If $k > k_{\max}$ go to 3
(i) If $t = d$ then multiply w_{k+1} by $(1 - w_d)$ and $v_{k+1}^{\max}$ by $(1 - v_d)$
(j) If stopping condition is satisfied then go to 3
(k) Increment i. If $i > p$ then increment k, and set $i = 1$
(l) Go to 2(a)

3. Report results
4. Terminate

The above logic is illustrated as a flow diagram in Figure 1 without detailing the working of the dynamic reduction parameters. Problem independent stopping conditions based on convergence tests are difficult to define for global optimizers. Consequently, we typically use a fixed number of fitness evaluations or swarm iterations as a stopping criteria.

PARALLEL PARTICLE SWARM ALGORITHM

The following issues had to be addressed in order to create a parallel PSO algorithm.

Concurrent operation and scalability

The algorithm should operate in such a fashion that it can be easily decomposed for parallel operation on a multi-processor machine. Furthermore, it is highly desireable that it be scalable. Scalability implies that the nature of the algorithm should not place a limit on the number of computational nodes that can be utilized, thereby permitting full use of available computational resources.

An example of an algorithm with limited scalability is a parallel implementation of a gradient-based optimizer. This algorithm is decomposed by distributing the workload of the derivative calculations for a single point in design space among multiple processors. The upper limit on concurrent operations using this approach is therefore set by the number of design variables in the problem.

On the other hand, population-based methods such as the GA and PSO are better suited to parallel computing. Here the population of individuals representing designs can be increased or decreased according to the availability and speed of processors. Any additional agents in the population will allow for a higher fidelity search in the design space, lowering susceptibility to entrapment in local minima. However, this comes at the expense of additional fitness evaluations.

Asynchronous vs synchronous implementation

The original PSO algorithm was implemented with a synchronized scheme for updating the best 'remembered' individual and group fitness values f_k^i and f_k^g, respectively, and their associated positions $\mathbf{p}_k^i$ and $\mathbf{p}_k^g$. This approach entails performing the fitness evaluations for the entire swarm before updating the best fitness values. Subsequent experimentation revealed that improved convergence rates can be obtained by updating the f_k^i and f_k^g values and their positions after each individual fitness evaluation (i.e. in an asynchronous fashion) [11, 12].

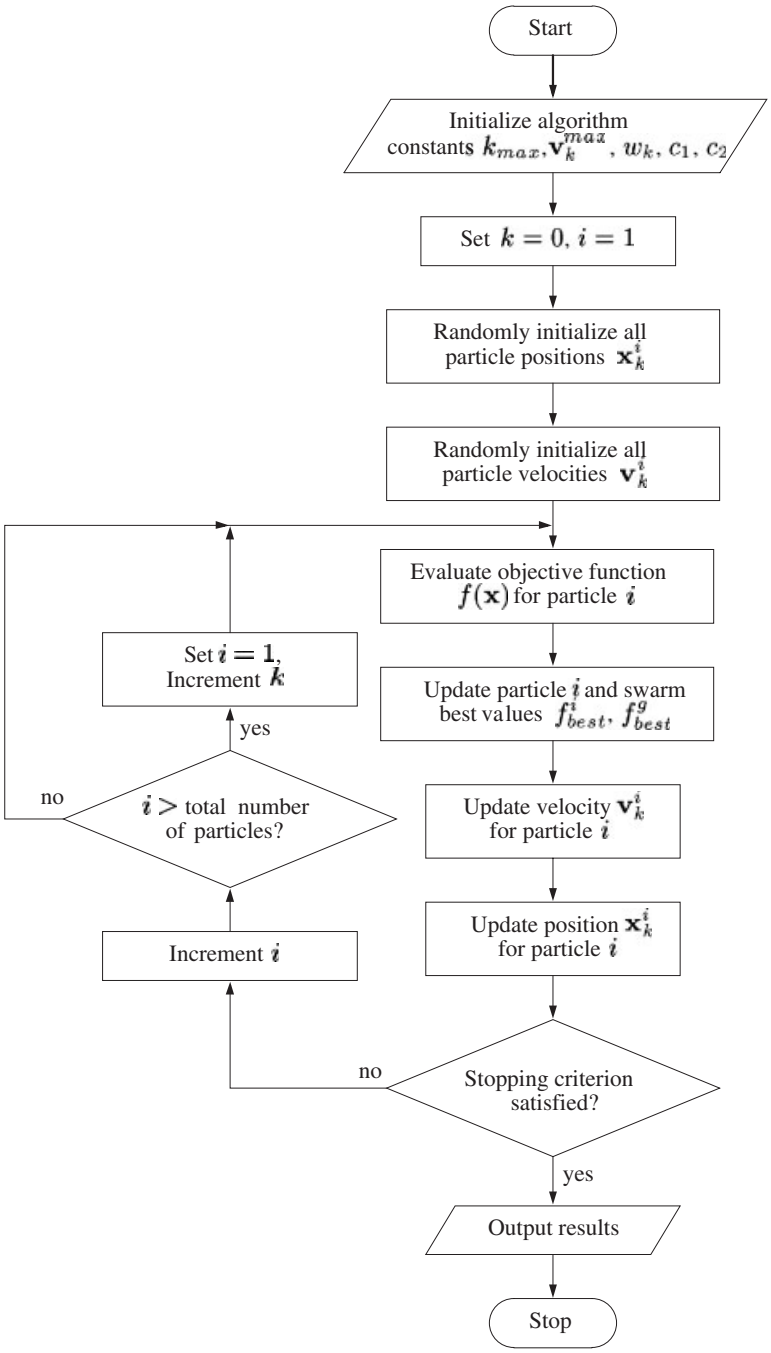

Figure 1. Serial implementation of the PSO algorithm. To avoid complicating the diagram, we have omitted velocity/inertia reduction operations.

It is speculated that because the updating occurs immediately after each fitness evaluation, the swarm reacts more quickly to an improvement in the best-found fitness value.

With the parallel implementation, however, this asynchronous improvement on the swarm is lost since fitness evaluations are performed concurrently. The parallel algorithm requires updating f_k^i and f_k^g for the entire swarm after all fitness evaluations have been performed, as in the original particle swarm formulation. Consequently, the swarm will react more slowly to changes of the best fitness value 'position' in the design space. This behaviour produces an unavoidable performance loss in terms of convergence rate compared to the asynchronous implementation and can be considered part of the overhead associated with parallelization.

Coherence

Parallelization should have no adverse affect on algorithm operation. Calculations sensitive to program order should appear to have occurred in exactly the same order as in the serial synchronous formulation, leading to the exact same final answer. In the serial PSO algorithm the fitness evaluations form the bulk of the computational effort for the optimization and can be performed independently. For our parallel implementation, we therefore chose a coarse decomposition scheme where the algorithm performs only the fitness evaluations concurrently on a parallel machine. Step 2 of the particle swarm optimization algorithm was modified accordingly to operate in a parallel manner:

2. Optimize
   (a) Update particle velocity vector $\mathbf{v}_{k+1}^i$ using Equation (2).
   (b) Update particle position vector $\mathbf{x}_{k+1}^i$ using Equation (1).
   (c) Concurrently evaluate fitness values f_k^i using design space co-ordinates $\mathbf{x}_{k+1}^i$ for $i = 1, \ldots, p$
   (d) If $f_{k+1}^i \leqslant f_{\text{best}}^i$ then $f_{\text{best}}^i = f_{k+1}^i$, $\mathbf{p}^{i+1} = \mathbf{x}_{k+1}^i$ for $i = 1, \ldots, p$
   (e) If $f_{k+1}^i \leqslant f_{\text{best}}^g$ then $f_{\text{best}}^g = f_{k+1}^i$, $\mathbf{p}_{k+1}^g = \mathbf{x}_{k+1}^i$ for $i = 1, \ldots, p$
   (f) If f_{best}^g was improved in (e) then reset $t = 0$. Else Increment t
   (g) If $k > k_{\max}$ go to 3
   (h) If $t = d$ then multiply w_{k+1} by $(1 - w_d)$ and $v_{k+1}^{\max}$ by $(1 - v_d)$
   (i) Increment k.
   (j) Go to 2(a).

The parallel PSO algorithm is represented by the flow diagram in Figure 2.

Network communication

In a parallel computational environment, the main performance bottleneck is often the communication latency between processors. This issue is especially relevant to large clusters of computers where the use of high performance network interfaces are limited due to their high cost. To keep communication between different computational nodes at a minimum, we use fitness evaluation tasks as the level of granularity for the parallel software. As previously mentioned, each of these evaluations can be performed independently and requires no communication aside from receiving design space co-ordinates to be evaluated and reporting the fitness value at the end of the analysis.

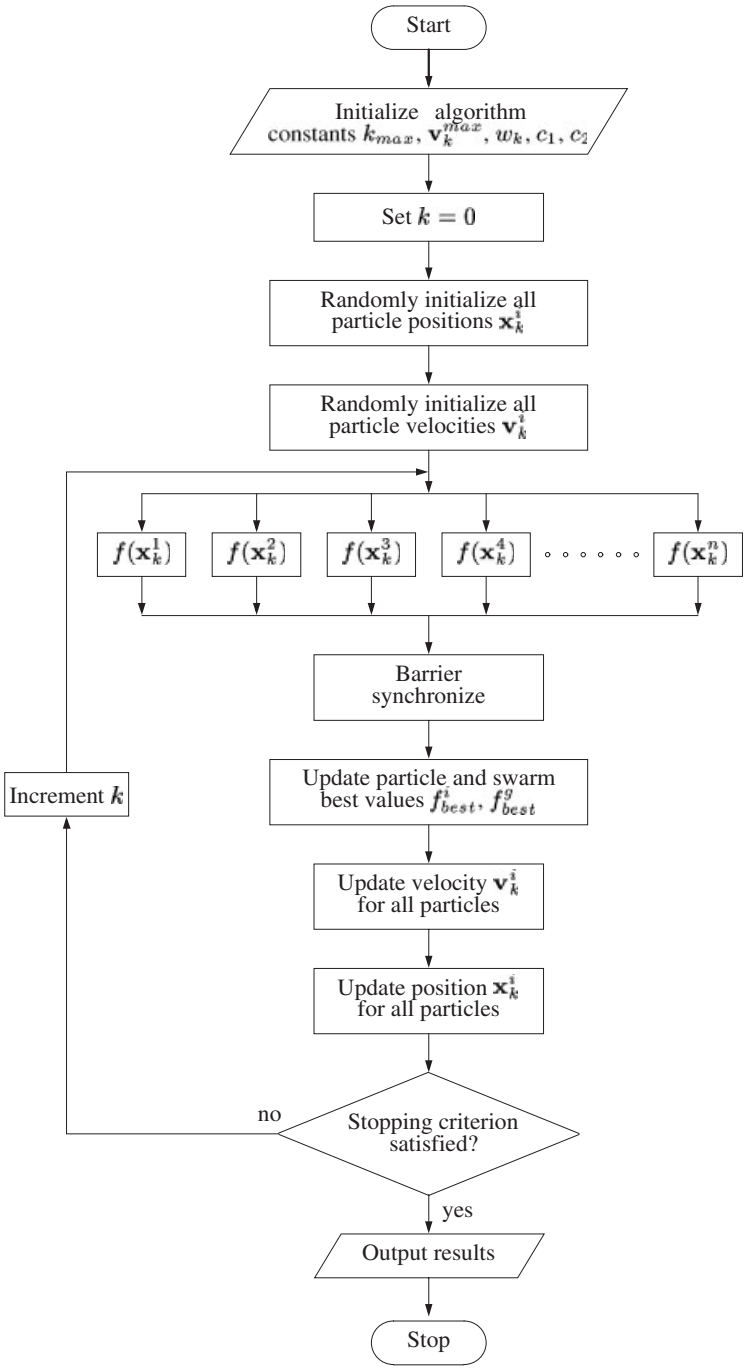

Figure 2. Parallel implementation of the PSO algorithm. We have again omitted velocity/inertia reduction operations to avoid complicating the diagram.

The optimization infrastructure is organized into a co-ordinating node and several computational nodes. PSO algorithm functions and task orchestration are performed by the co-ordinating node, which assigns the design co-ordinates to be evaluated, in parallel, to the computational nodes. With this approach, no communication is required between computational nodes as individual fitness evaluations are independent of each other. The only necessary communication is between the co-ordinating node and the computational nodes and encompasses passing the following information:

1. Several distinct design variable configuration vectors assigned by co-ordinating node to slave nodes for fitness evaluation.
2. Fitness values reported from slave nodes to co-ordinating node.
3. Synchronization signals to maintain program coherence.
4. Termination signals from co-ordinating node to slave nodes on completion of analysis to stop the program cleanly.

The parallel PSO scheme and required communication layer were implemented in ANSI C on a Linux operating system using the message passing interface (MPI) libraries.

Synchronization and implementation

From the parallel PSO algorithm, it is clear that some means of synchronization is required to ensure that all of the particle fitness evaluations have been completed and results reported before the velocity and position calculations can be executed (steps 2a and 2b). Synchronization is done using a barrier function in the MPI communication library which temporarily stops the co-ordinating node from proceeding with the next swarm iteration until all of the computational nodes have responded with a fitness value. Because of this approach, the time required to perform a single parallel swarm fitness evaluation will be dictated by the slowest fitness evaluation in the swarm.

Two networked clusters of computers were used to obtain the numerical results. The first cluster was used to solve the analytical test problems and comprised 40 1.33 GHz Athlon PCs located in the High-performance Computing and Simulation (HCS) Research Laboratory at the University of Florida. The second group was used to solve the biomechanical system identification problems and consisted of 32 2.40 GHz Intel PCs located in the HCS Research Laboratory at Florida State University. In both locations, 100 Mbps switched networks were utilized for connecting nodes.

SAMPLE OPTIMIZATION PROBLEMS

Analytical test problems

Two well-known analytical test problems were used to evaluate parallel PSO algorithm performance on large-scale problems with multiple local minima (see appendix for mathematical description of both problems). The first was a test function (Figure 3(a)) introduced by Griewank [21] which superimposes a high-frequency sine wave on a multi-dimensional parabola. In contrast, the second problem used the Corana test function [22] which exhibits discrete jumps throughout the design space (Figure 3(b)). For both problems, the number of local minima increases exponentially with the number of design variables. To investigate large-scale optimization

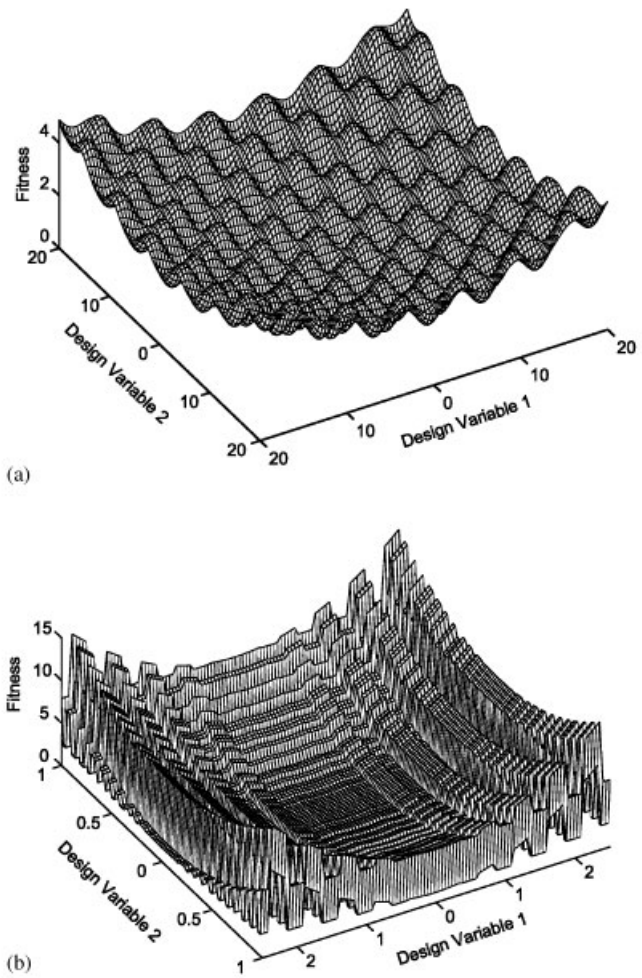

Figure 3. Surface plots of the; (a) Griewank; and (b) Corana analytical test problems showing the presence of multiple local minima. For both plots, 126 design variables were fixed at their optimal values and the remaining 2 design variables varied in a small region about the global minimum.

issues, we formulated both problems using 128 design variables. Since fitness evaluations are extremely fast for these test problems, a delay of approximately half a second was built into each fitness evaluation so that total computation time would not be swamped by communication time.

Since parallelization opens up the possibility of utilizing large numbers of processors, we used the analytical test problems to investigate how convergence rate and final solution are affected by the number of particles employed in a parallel PSO run. To ensure that all swarms were given equally 'fair' starting positions, we generated a pool of 128 initial positions using the latin hypercube sampler (LHS). Particle positions selected with this scheme will be distributed uniformly throughout the design space [23].

This initial pool of 128 particles was divided into the following sub-swarms: one swarm of 128 particles, two swarms of 64 particles, four swarms of 32 particles, and eight swarms of 16 particles. Each sub-swarm was used independently to solve the two analytical test problems. This approach allowed us to investigate whether is it more efficient to perform multiple parallel optimizations with smaller population sizes or one parallel optimization with a larger population size given a sufficient number of processors. To obtain comparisons for convergence speed, we allowed all PSO runs to complete 10 000 iterations before the search was terminated. This number of iterations corresponded to between 160 000 and 1 280 000 fitness evaluations depending on the number of particles employed in the swarm.

Biomechanical system identification problems

In addition to the analytical test problems, medium-scale biomechanical system identification problems were used to evaluate parallel PSO performance under more realistic conditions. These problems were variations of a general problem that attempts to find joint parameters (i.e. positions and orientations of joint axes) that match a kinematic ankle model to experimental surface marker data [24]. The data are collected with an optoelectronic system that uses multiple cameras to record the positions of external markers placed on the body segments. To permit measurement of three-dimensional motion, we attach three non-colinear markers to the foot and lower leg. The recordings are processed to obtain marker trajectories in a laboratory-fixed co-ordinate system [25, 26]. The general problem possesses 12 design variables and requires approximately 1 minute for each fitness evaluation. Thus, while the problem is only medium-scale in terms of number of design variables, it is still computationally costly due to the time required for each fitness evaluation.

The first step in the system identification procedure is to formulate a parametric ankle joint model that can emulate a patient's movement by possessing sufficient degrees of freedom. For the purpose of this paper, we approximate the talocrural and subtalar joints as simple 1 degree of freedom revolute joints. The resulting ankle joint model (Figure 4) contains 12 adjustable parameters that define its kinematic structure [24]. The model also has a set of virtual markers fixed to the limb segments in positions corresponding to the locations of real markers on the subject. The linkage parameters are then adjusted via optimization until markers on the model follow the measured marker trajectories as closely as possible.

To quantify how closely the kinematic model with specified parameter values can follow measured marker trajectories, we define a cumulative marker error e as follows:

$$e = \sum_{j=1}^{n} \sum_{i=1}^{m} \Delta_{i,j}^2 \tag{3}$$

where

$$\Delta_{i,j}^2 = \Delta x_{i,j}^2 + \Delta y_{i,j}^2 + \Delta z_{i,j}^2 \tag{4}$$

where $\Delta x_{i,j}$, $\Delta y_{i,j}$ and $\Delta z_{i,j}$ are the spatial displacement errors for marker i at time frame j in the x, y, and z directions as measured in the laboratory-fixed co-ordinate system, $n = 50$ is the number of time frames, and $m = 6$ (3 on the lower leg and 3 on the foot) is the number of markers. These errors are calculated between the experimental marker locations on the human subject and the virtual marker locations on the kinematic model. For each time

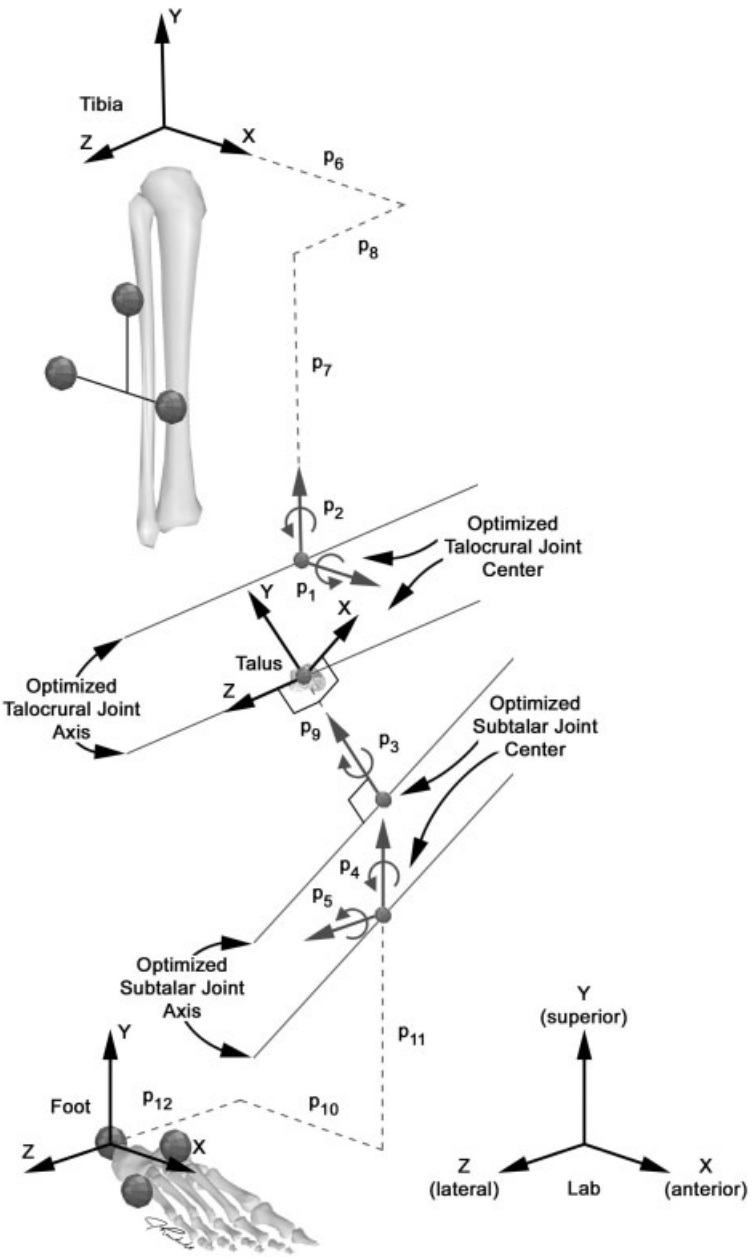

Figure 4. Joint locations and orientations in the parametric ankle kinematic model. Each p_i $(i = 1, \ldots, 12)$ represents a different position or orientation parameter in the model.

frame, a non-linear least squares sub-optimization is performed to determine the joint angles that minimize $\Delta_{i,j}^2$ given the current set of model parameters. The first sub-optimization is started from an initial guess of zero for all joint angles. The sub-optimization for each subsequent time frame is started with the solution from the previous time frame to speed convergence. By performing a separate sub-optimization for each time frame and then calculating the sum of the squares of the marker co-ordinate errors, we obtain an estimate of how well the model fits the data for all time frames included in the analysis. By varying the model parameters and repeating the sub-optimization process, the parallel PSO algorithm finds the best set of model parameters that minimize e over all time frames.

For numerical testing, three variations of this general problem were analysed as described below. In all cases the number of particles used by the parallel PSO algorithm was set to a recommended value of 20 [12].

1. *Synthetic data without numerical noise*: Synthetic (i.e. computer generated) data without numerical noise were generated by simulating marker movements using a lower body kinematic model with virtual markers. The synthetic motion was based on an experimentally measured ankle motion (see 3 below). The kinematic model used anatomically realistic joint positions and orientations. Since the joint parameters associated with the synthetic data were known, this optimization was used to verify that the parallel PSO algorithm could accurately recover the original model.
2. *Synthetic data with numerical noise*: Numerical noise was superimposed on each synthetic marker co-ordinate trajectory to emulate the effect of marker displacements caused by skin movement artefacts [27]. A previously published noise model requiring three random parameters was used to generate a perturbation N in each marker co-ordinate [28]:

$$N = A\ \sin(\omega t + \phi) \tag{5}$$

   where A is the amplitude, ω the frequency, and ϕ the phase angle of the noise. These noise parameters were treated as uniform random variables within the bounds $0 \leqslant A \leqslant 1\text{cm}$, $0 \leqslant \omega \leqslant 25\ \text{rad/s}$, and $0 \leqslant \phi \leqslant 2\pi$ [28].
3. *Experimental data*: Experimental marker trajectory data were obtained by processing three-dimensional recordings from a subject performing movements with reflective markers attached to the foot and lower leg as previously described. Institutional review board approval was obtained for the experiments and data analysis, and the subject gave informed consent prior to participation. Marker positions were reported in a laboratory-fixed co-ordinate system.

Speedup and parallel efficiency

Parallel performance for both classes of problems was quantified by calculating speedup and parallel efficiency for different numbers of processors. Speedup is the ratio of sequential execution time to parallel execution time and ideally should equal the number of processors. Parallel efficiency is the ratio of speedup to number of processors and ideally should equal 100%. For the analytical test problems, only the Corana problem was run since the half second delay added to both problems makes their parallel performance identical. For the biomechanical system identification problems, only the synthetic data with numerical noise case was reported since experimentation with the other two cases produced similar parallel performance.

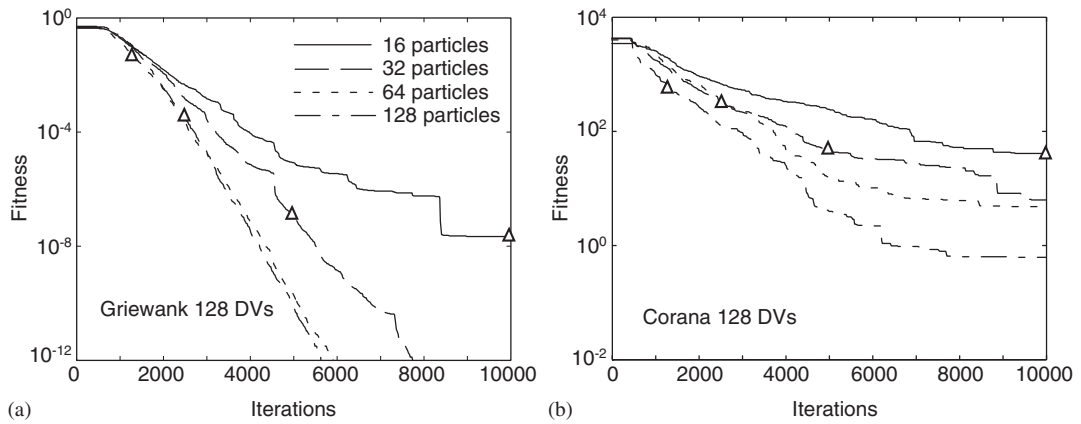

Figure 5. Average fitness convergence histories for the; (a) Griewank; and (b) Corana analytical test problems for swarm sizes of 16, 32, 64, and 128 particles and 10000 swarm iterations. Triangles indicate the location on each curve where 160 000 fitness evaluations were completed.

The number of particles and nodes used for each parallel evaluation was selected based on the requirements of the problem. The Corana problem with 128 design variables was solved using 32 particles and 1, 2, 4, 8, 16, and 32 nodes. The biomechanical problem with 12 design variables was solved using 20 particles and 1, 2, 5, 10, and 20 nodes. Both problems were allowed to run until 1000 fitness evaluations were completed.

NUMERICAL RESULTS

Convergence rates for the two analytical test problems differed significantly with changes in swarm size. For the Griewank problem (Figure 5(a)), individual PSO runs converged to within 1e-6 of the global minimum after 10 000 optimizer iterations, regardless of the swarm size. Run-to-run variations in final fitness value (not shown) for a fixed swarm size were small compared to variations between swarm sizes. For example, no runs with 16 particles produced a better final fitness value than any of the runs with 32 particles, and similarly for the 16–32, 32–64, and 64–128 combinations. When number of fitness evaluations was considered instead of number of swarm iterations, runs with a smaller swarm size tended to converge more quickly than did runs with a larger swarm size (see triangles in Figure 5). However, two of the eight runs with the smallest number of particles failed to show continued improvement near the maximum number of iterations, indicating possible entrapment in a local minimum. Similar results were found for the Corana problem (Figure 5(b)) with two exceptions. First, the optimizer was unable obtain the global minimum for any swarm size within the specified number of iterations (Figure 5(b)), and second, overlapping in results between different swarm sizes was observed. For example, some 16 particle results were better than 32 particles results, and similarly for the other neighboring combinations. On average, however, a larger swarm size tended to produce better results for both problems.

Table I. Parallel PSO results for the biomechanical system identification problem using synthetic marker trajectories without and with numerical noise.

Model parameter	Upper bound	Lower bound	Synthetic solution	Synthetic data	
				Without noise	With noise
p_1 (deg)	48.67	−11.63	18.37	18.36	15.13
p_2 (deg)	30.00	−30.00	0.00	− 0.01	8.01
p_3 (deg)	70.23	10.23	40.23	40.26	32.97
p_4 (deg)	53.00	− 7.00	23.00	23.03	23.12
p_5 (deg)	72.00	12.00	42.00	42.00	42.04
p_6 (cm)	6.27	− 6.27	0.00	0.00	− 0.39
p_7 (cm)	−33.70	−46.24	−39.97	−39.97	−39.61
p_8 (cm)	6.27	− 6.27	0.00	− 0.00	0.76
p_9 (cm)	0.00	− 6.27	− 1.00	− 1.00	− 2.82
p_{10} (cm)	15.27	2.72	9.00	9.00	10.21
p_{11} (cm)	10.42	− 2.12	4.15	4.15	3.03
p_{12} (cm)	6.89	− 5.65	0.62	0.62	− 0.19

Both optimizations with 20 particles were terminated after 40 000 fitness evaluations.

Table II. Synthetic and experimental marker distance and joint parameter RMS errors for the biomechanical system identification problem.

RMS errors	Synthetic Data		Experimental data
	Without noise	With noise	
Marker distances (cm)	3.58e-4	0.568	0.394
Orientation parameters (deg)	1.85e-2	5.01	N/A
Position parameters (cm)	4.95e-4	1.00	N/A

The parallel PSO algorithm found ankle joint parameters consistent with the known solution or results in the literature [24]. The algorithm had no difficulty recovering the original parameters from the synthetic date set without noise (Table I), producing a final cumulative error e on the order of 10^{-13}. The original model was recovered with mean orientation errors less than 0.05° and mean position errors less than 0.008 cm. Furthermore, the parallel implementation produced identical fitness and parameter histories as did a synchronous serial implementation. For the synthetic data set with superimposed noise, a RMS marker distance error of 0.568 cm was found, which is on the order of the imposed numerical noise with maximum amplitude of 1 cm. For the experimental data set, the RMS marker distance error was 0.394 cm (Table II), comparable to the error for the synthetic data with noise.

Convergence characteristics were similar for the three data sets considered in this study. The initial convergence rate was quite high (Figure 6(a)), whereafter it slowed when the approximate location of the global minimum was found. As the solution process proceeded, the optimizer traded off increases in RMS joint orientation error (Figure 6(b)) for decreases in RMS joint position error (Figure 6(c)) to achieve further minor reductions in the fitness value.

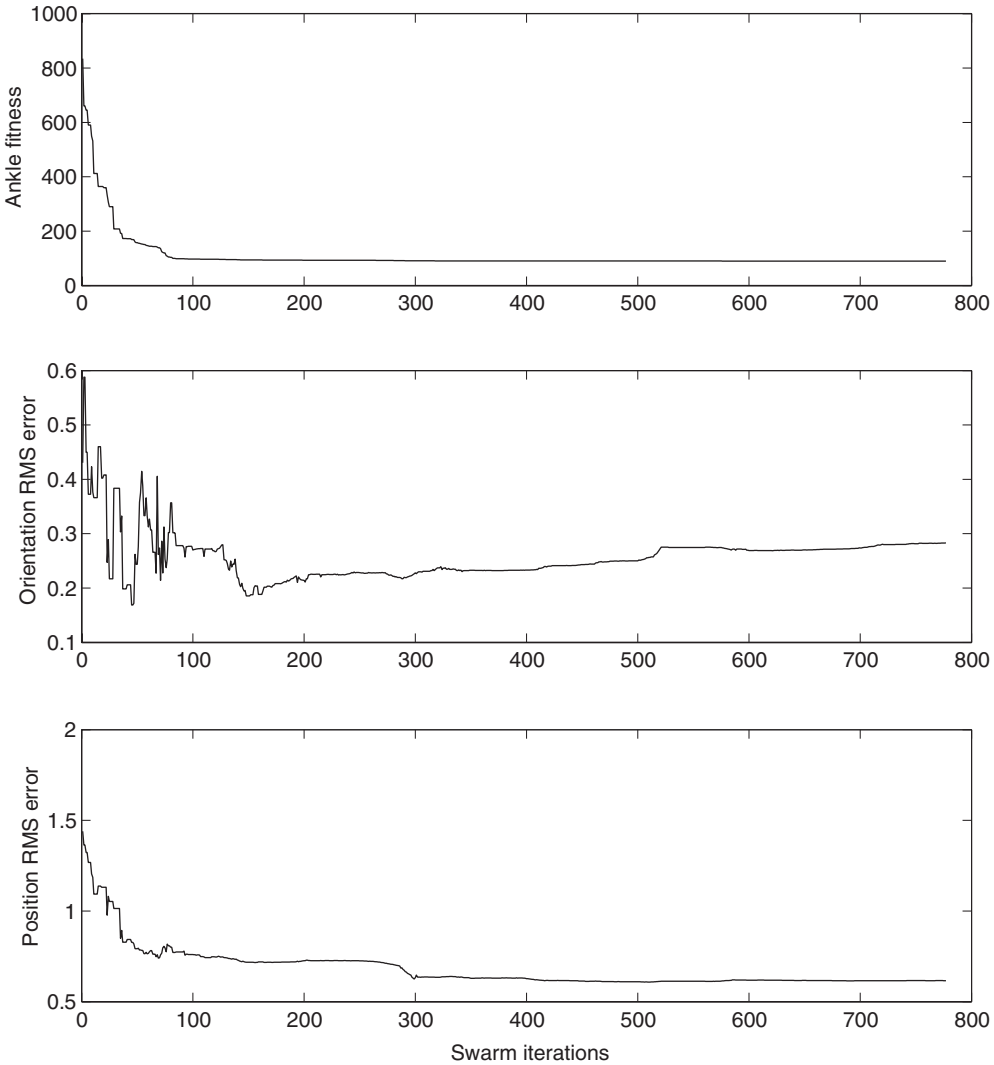

Figure 6. Fitness convergence and parameter error plots for the biomechanical system identification problem using synthetic data with noise.

The analytical and biomechanical problems exhibited different parallel performance characteristics. The analytical problem demonstrated almost perfectly linear speedup (Figure 7(a), squares) resulting in parallel efficiencies above 95% for up to 32 nodes (Figure 7(b), squares). In contrast, the biomechanical problem exhibited speedup results that plateaued as the number of nodes was increased (Figure 7(a), circles), producing parallel efficiencies that decreased almost linearly with increasing number of nodes (Figure 7(b), circles). Each additional node produced roughly a 3% reduction in parallel efficiency.

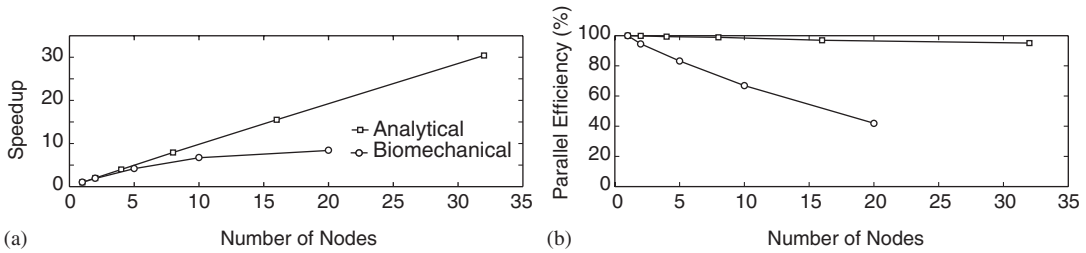

Figure 7. (a) Speedup and (b) parallel efficiency for the analytical and biomechanical optimization problems.

DISCUSSION

This study presented a parallel implementation of the particle swarm global optimizer. The implementation was evaluated using analytical test problems and biomechanical system identification problems. Speedup and parallel efficiency results were excellent when each fitness evaluation took the same amount of time. For problems with large numbers of design variables and multiple local minima, maximizing the number of particles produced better results than repeated runs with fewer particles. Overall, parallel PSO makes efficient use of computational resources and provides a new option for computationally demanding engineering optimization problems.

The agreement between optimized and known orientation parameters $p_1 - p_4$ for the biomechanical problem using synthetic data with noise was poorer than initially expected. This finding was the direct result of the sensitivity of orientation calculations to errors in marker positions caused by the injected numerical noise. Because of the close proximity of the markers to each other, even relatively small amplitude numerical noise in marker positions can result in large fluctuations in the best-fit joint orientations. While more time frames could be used to offset the effects of noise, this approach would increase the cost of each fitness evaluation due to an increased number of sub-optimizations. Nonetheless, the fitness value for the optimized parameters was lower than that for the parameters used to generate the original noiseless synthetic data.

Though the biomechanical optimization problems only involved 12 design variables, multiple local minima existed when numerical or experimental noise was present. When the noisy synthetic data set was analyzed with a gradient-based optimizer using 20 random starting points, the optimizer consistently found distinct solutions, indicating a large number of local minima. Similar observations were made for a smaller number of gradient-based runs performed on the experimental data set. To evaluate the parallel PSOs ability to avoid entrapment in these local minima, we performed 10 additional runs with the algorithm. All 10 runs converged to the same solution, which was better than any of the solutions found by gradient-based runs.

Differences in parallel PSO performance between the analytical test problem and the biomechanical system identification problem can be explained by load balancing issues. The half second delay added to the analytical test problem made all fitness evaluations take approximately the same amount of time and substantially less time than communication tasks. Consequently, load imbalances were avoided and little degradation in parallel performance was observed with increasing number of processors. In contrast, for the biomechanical system identification

problem, the time required to complete the 50 sub-optimizations was sensitive to the selected point in design space, thereby producing load imbalances. As the number of processors increased, so did the likelihood that at least one fitness evaluation would take much longer than the others. Due to the synchronization requirement of the current parallel implementation, the resulting load imbalance caused by even one slow fitness evaluation was sufficient to degrade parallel performance rapidly with increasing number of nodes. An asynchronous parallel implementation could be developed to address this problem with the added benefit of permitting high parallel efficiency on inhomogeneous clusters.

Our results for the analytical and biomechanical optimization problems suggest that PSO performs best on problems with continuous rather than discrete noise. The algorithm consistently found the global minimum for the Griewank problem, even when the number of particles was low. Though the global minimum is unknown for the biomechanical problem using synthetic data with noise, multiple PSO runs consistently converged to the same solution. Both of these problems utilized continuous, sinusoidal noise functions. In contrast, PSO did not converge to the global minima for the Corana problem with its discrete noise function. Thus, for large-scale problems with multiple local minima and discrete noise, other optimization algorithms such as GA may provide better results [5].

Use of a LHS rather than uniform random sampling to generate initial points in design space may be a worthwhile PSO algorithm modification. Experimentation with our random number generator indicated that initial particle positions can at times be grouped together. This motivated our use of LHS to avoid re-sampling the same region of design space when providing initial guesses to sub-swarms. To investigate the influence of sampling method on PSO convergence rate, we performed multiple runs with the Griewank problem using uniform random sampling and a LHS with the default design variable bounds (-600 to $+600$) and with the bounds shifted by 200 (-400 to $+800$). We found that when the bounds were shifted, convergence rate with uniform random sampling changed while it did not with a LHS. Thus, swarm behaviour appears to be influenced by sampling method, and a LHS may be helpful for minimizing this sensitivity.

A secondary motivation for running the analytical test problems with different numbers of particles was to determine whether the use of sub-swarms would improve convergence. The question is whether a larger swarm where all particles communicate with each other is more efficient than multiple smaller swarms where particles communicate within each sub-swarm but not between sub-swarms. It is possible that the global best position found by a large swarm may unduly influence the motion of all particles in the swarm. Creating sub-swarms that do not communicate eliminates this possibility. In our approach, we performed the same number of fitness evaluations for each population size. Our results for both analytical test problems suggest that when a large numbers of processors is available, increasing the swarm size will increase the probability of finding a better solution.

Analysis of PSO convergence rate for different numbers of particles also suggests an interesting avenue for future investigation. Passing an imaginary curve through the triangles in Figure 5 reveals that for a fixed number of fitness evaluations, convergence rate increases asymptotically with decreasing number of particles. While the solution found by a smaller number of particles may be a local minimum, the final particle positions may still identify the general region in design space where the global minimum is located. Consequently, an adaptive PSO algorithm that periodically adjusts the number of particles upward during the course of an optimization may improve convergence speed. For example, an initial run with 16 particles

could be performed for a fixed number of fitness evaluations. At the end of that phase, the final positions of those 16 particles would be kept, but 16 new particles would be added to bring the total up to 32 particles. The algorithm would continue using 32 particles until the same number of fitness evaluations was completed. The process of gradually increasing the number of particles would continue until the maximum specified swarm size (e.g. 128 particles) was analysed. To ensure systematic sampling of the design space, a LHS would be used to generate a pool of sample points equal to the maximum number of particles and from which sub-samples would be drawn progressively at each phase of the optimization. In the scenario above with a maximum of 128 particles, the first phase with 16 particles would remove 16 sampled points from the LHS pool, the second phase another 16 points, the third phase 32 points, and the final phase the remaining 64 points.

CONCLUSIONS

In summary, the parallel Particle Swarm Optimization algorithm presented in this study exhibits excellent parallel performance as long as individual fitness evaluations require the same amount of time. For optimization problems where the time required for each fitness evaluation varies substantially, an asynchronous implementation may be needed to reduce wasted CPU cycles and maintain high parallel efficiency. When large numbers of processors are available, use of larger population sizes may result in improved convergence rates to the global solution. An adaptive PSO algorithm that increases population size incrementally may also improve algorithm convergence characteristics.

APPENDIX A

A.1. Analytical test problems

A.1.1. Modified Griewank

Objective function:

$$f(\mathbf{x}) = \sum_{i=1}^{n} x_i^2/d - \prod_{i=1}^{n} \cos(x_i/\sqrt{i}) + 1$$

with $n = 128$ and $d = 4000$.

Search domain:

$$D = \{(x_1, x_2, \ldots, x_{128}) \in \Re^{128} : -600.0 \leqslant x_i \leqslant 600.0,\ \ i = 1, 2, \ldots, 128\}.$$

Solution:

$$\mathbf{x}_i^* = 0.0, \quad i = 1, \ldots, 128, \quad f^* = 0.0$$

A.1.2. Modified Corana

Objective function:

$$f(\mathbf{x}) = \sum_{i=1}^{n} \begin{cases} (t\ \mathrm{sgn}(z_i) + z_i)^2 cd & \text{if } |x_i - z_i| < t \\ d_i x_i^2 & \text{otherwise} \end{cases}$$

where $n = 128$ and

$$z_i = \left\lfloor \left| \frac{x_i}{s} \right| + 0.49999 \right\rfloor \operatorname{sgn}(x_i)\, s$$

$$c = 0.15$$

$$s = 0.2$$

$$t = 0.05$$

$$d_i = \begin{cases} 1 & i = 1, 5, 9, \ldots \\ 1000 & i = 2, 6, 10, \ldots \\ 10 & i = 3, 7, 11, \ldots \\ 100 & i = 4, 8, 12, \ldots \end{cases}$$

Search domain:

$$D = \{(x_1, x_2, \ldots, x_{128}) \in \Re^{128} : -1000.0 \leqslant x_i \leqslant 1000.0, \;\; i = 1, 2, \ldots, 128\}$$

Solution:

$$|\mathbf{x}_i^*| \leqslant 0.05, \quad i = 1, \ldots, 128, \quad f^* = 0.0$$

ACKNOWLEDGEMENTS

The authors gratefully acknowledge funding for this study from NIH National Library of Medicine (R03 LM07332) and Whitaker Foundation grants to B. J. Fregly and an AFOSR (F49620-09-1-0070) grant to R. T. Haftka.

REFERENCES

1. Snir M, Otto S, Huss-Lederman S, Walker D, Dongarra J. *MPI*: *The Complete Reference*. Massachusetts Institute of Technology: Cambridge, MA, 1996.
2. Gropp W, Lusk E. *User's Guide for mpich, A Portable Implementation of MPI*. Argonne National Laboratory, Mathematics and Computer Science Division, http:// www.mcs.anl.gov/mpi/mpiuserguide/paper.html.
3. Geist A, Beguelin A, Dongarra J, Manchek R, Jiang W, Sunderam V. PVM 3 user's guide and reference manual. *Technical Report ORNL/TM-12187*, Oak Ridge National Laboratory, Knoxville, TN, 1994.
4. Anderson FC, Ziegler J, Pandy MG, Whalen RT. Application of high-performance computing to numerical simulation of human movement. *Journal of Biomechanical Engineering* 1995; **117**:300–308.
5. van Soest AJK, Casius LJR. The merits of a parallel genetic algorithm in solving hard optimization problems. *Journal of Biomechanical Engineering* 2003; **125**:141–146.
6. Monien B, Ramme F, Salmen H. A parallel simulated annealing algorithm for generating 3D layouts of undirected graphs. In *Proceedings of the 3rd International Symposium on Graph Drawing*, *GD*, Franz JB (ed.). Springer: Berlin, Germany, 1995; 396–408.
7. Eberhart RC, Shi Y. Particle swarm optimization: developments, applications, and resources. In *Proceedings of the 2001 Congress on Evolutionary Computation CEC2001*, Seoul, Korea. IEEE Press: New York, 27–30 May 2001; 81–86.

8. Venter G, Sobieszczanski-Sobieski J. Multidisciplinary optimization of a transport aircraft wing using particle swarm optimization. In *9th AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Atlanta, GA, 2002.
9. Fourie PC, Groenwold AA. Particle swarms in algorithm in topology optimization. In *Proceedings of Fourth World Congress of Structural and Multidisciplinary Optimization*, Dalian, China, May 2001; 52–53.
10. Eberhart RC, Shi Y. Particle swarm optimization: developments, applications and resources. In *Proceedings of the IEEE Congress on Evolutionary Computation* (*CEC 2001*), Korea. IEEE Press: New York, 27–30 May 2001.
11. Schutte JF. Particle swarms in sizing and global optimization. *Master's Thesis*, University of Pretoria, Department of Mechanical Engineering, 2002.
12. Carlisle A, Dozier G. An off-the-shelf pso. In *Proceedings of the Workshop on Particle Swarm Optimization*. Purdue School of Engineering and Technology, Indianapolis, USA, 2001.
13. Kennedy J, Eberhart RC. Particle swarm optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, vol. 4, Perth, Australia. IEEE Service Center: Piscataway, NJ, 1995; 1942–1948.
14. Shi Y, Eberhart RC. A modified particle swarm optimizer. In *Proceedings of the IEEE International Conference on Evolutionary computation*, Anchorage, Alaska. IEEE Press: Piscataway, USA, 1998; 69–73.
15. Shi Y, Eberhart RC. Parameter selection in particle swarm optimization. In *Evolutionary Programming VII*, Porto VW, Saravanan N, Waagen D, Eiben AE (eds). Lecture Notes in Computer Science, vol. 1447. Springer: Berlin, 1998; 591–600.
16. Eberhart RC, Shi Y. Comparing inertia weights and constriction factors in particle swarm optimization. In *Proceedings of the 2000 Congress on Evolutionary Computation*. IEEE Service Center: Piscataway, NJ, 2000; 84–88.
17. Clerc M. The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. In *Proceedings of the Congress of Evolutionary Computation*, Angeline PJ, Michalewicz Z, Schoenauer M, Yao X, Zalzala A (eds), vol. 3, Washington DC, USA. IEEE Press: New York, 6–9 July 1999; 1951–1957.
18. Løvbjerg M, Rasmussen TK, Krink T. Hybrid particle swarm optimiser with breeding and subpopulations. In *Proceedings of the Third Genetic and Evolutionary Computation Conference* (*GECCO-2001*), San Francisco, CA, 2001.
19. Carlisle A, Dozier G. Adapting particle swarm optimization to dynamic environments. In *International Conference on Artificial Intelligence*, vol. I. Las Vegas, NV, 2000; 429–434.
20. Kennedy J, Eberhart RC. A discrete binary version of the particle swarm algorithm. In *Proceedings of the 1997 Conference on Systems, Man and Cybernetics*. IEEE Service Center: Piscataway, NJ, 1997; 4104–4109.
21. Griewank AO. Generalized descent for global optimization. *Journal of Optimization Theory and Application* 1981; **34**:11–39.
22. Corana A, Marchesi M, Martini C, Ridella S. Minimizing multimodal functions of continuous variables with the 'simulated annealing' algorithm. *ACM Transactions in Mathematical Software* 1987; **13**(3):262–280.
23. Wyss GD, Jorgensen KH. A user's guide to lhs: Sandia's latin hypercube sampling software. *Sandia National Laboratories Technical Report SAND98-0210*, Albuquerque, NM, 1998.
24. van den Bogert AJ, Smith GD, Nigg BM. In vivo determination of the anatomical axes of the ankle joint complex: an optimization approach. *Journal of Biomechanics* 1994; **12**:1477–1488.
25. Soderkvist I, Wedin PA. Determining the movements of the skeleton using well-configured markers. *Journal of Biomechanics* 1993; **26**:1473–1477.
26. Spoor CW, Veldpaus FE. Rigid body motion calculated from spatial co-ordinates of markers. *Journal of Biomechanics* 1980; **13**:391–393.
27. Lu T-W, O'Connor JJ. Bone position estimation from skin marker co-ordinates using global optimization with joint constraints. *Journal of Biomechanics* 1999; **32**:129–134.
28. Cheze L, Fregly BJ, Dimnet J. A solidification procedure to facilitate kinematics analyses based on video system data. *Journal of Biomechanics* 1995; **28**:879–884.

