Published in final edited form as:
Int J Numer Methods Eng. 2006 July 23; 67(4): 578–595.

Parallel asynchronous particle swarm optimization

Byung-Il Koh[1], **Alan D. George**[1], **Raphael T. Haftka**[2], and **Benjamin J. Fregly**[2,3,*,†]
1 *Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL, U.S.A.*

2 *Department of Mechanical and Aerospace Engineering, University of Florida, Gainesville, FL, U.S.A.*

3 *Department of Biomedical Engineering, University of Florida, Gainesville, FL, U.S.A.*

SUMMARY

The high computational cost of complex engineering optimization problems has motivated the development of parallel optimization algorithms. A recent example is the parallel particle swarm optimization (PSO) algorithm, which is valuable due to its global search capabilities. Unfortunately, because existing parallel implementations are synchronous (PSPSO), they do not make efficient use of computational resources when a load imbalance exists. In this study, we introduce a parallel asynchronous PSO (PAPSO) algorithm to enhance computational efficiency. The performance of the PAPSO algorithm was compared to that of a PSPSO algorithm in homogeneous and heterogeneous computing environments for small- to medium-scale analytical test problems and a medium-scale biomechanical test problem. For all problems, the robustness and convergence rate of PAPSO were comparable to those of PSPSO. However, the parallel performance of PAPSO was significantly better than that of PSPSO for heterogeneous computing environments or heterogeneous computational tasks. For example, PAPSO was 3.5 times faster than was PSPSO for the biomechanical test problem executed on a heterogeneous cluster with 20 processors. Overall, PAPSO exhibits excellent parallel performance when a large number of processors (more than about 15) is utilized and either (1) heterogeneity exists in the computational task or environment, or (2) the computation-to-communication time ratio is relatively small.

Keywords

particle swarm; global optimization; parallel asynchronous algorithms; cluster computing

INTRODUCTION

Particle swarm optimization (PSO) is a stochastic optimization method often used to solve complex engineering problems such as structural and biomechanical optimizations (e.g. References [1–7]). Similar to evolutionary optimization methods, PSO is a derivative-free, population-based global search algorithm. PSO uses a population of solutions, called particles, which fly through the search space with directed velocity vectors to find better solutions. These velocity vectors have stochastic components and are dynamically adjusted based on historical and inter-particle information.

*Correspondence to: Benjamin J. Fregly, Department of Mechanical and Aerospace Engineering, 231 MAE-A Building, Box 116250, University of Florida, Gainesville, FL 32611-6250, U.S.A..
† fregly@ufl.edu

Contract/grant sponsor: NIH National Library of Medicine; contract/grant number: R03 LM07332 Contract/grant sponsor: Whitaker Foundation

Recent advances in computer and network technologies have led to the development of a variety of parallel optimization algorithms, most of which are synchronous in nature. A synchronous optimization algorithm is one that requires a synchronization point at the end of each optimizer iteration before continuing to the next iteration. For example, parallel population-based algorithms such as genetic [7–9], simulated annealing [10,11], pattern search [12], and particle swarm [5,6] methods typically wait for completion of all function evaluations by the population before determining a new set of design variables. Similarly, parallel gradient-based algorithms [13–16] require gradients to be calculated for all design variables before determining a new search direction.

Parallel synchronous optimization algorithms work best when three conditions are met. First, the optimization has total and undivided access to a homogeneous cluster of computers without interruptions from other users. Second, the analysis function takes a constant amount of time to evaluate any set of design variables throughout the optimization. Third, the number of parallel tasks can be equally distributed among the available processors. If any of these three conditions is not met, the parallel optimization algorithm will not make the most efficient use of the available computational resources. Since large clusters are often heterogeneous with multiple users, the first condition is frequently violated. Furthermore, the computational cost to evaluate a complex analysis function is often variable during an optimization [7]. Finally, since most parallel optimization algorithms have coarse granularity, the number of parallel tasks to be assigned to the available processors is usually not large enough to balance the workload. All three factors can lead to load imbalance problems that significantly degrade parallel performance.

Parallel optimization algorithms employing an asynchronous approach are a potential solution to the load imbalance problem. The asynchronous approach does not need a synchronization point to determine a new search direction or new sets of design variables. Thus, the optimization can proceed to the next iteration without waiting for the completion of all function evaluations from the current iteration. Despite this advantage, few parallel optimization algorithms have been asynchronous. Exceptions include specific parallel asynchronous implementations of Newton or quasi-Newton [17,18], pattern search [19], genetic [20,21], and simulated annealing [22] algorithms.

In this study, we propose and evaluate a parallel asynchronous particle swarm optimization (PAPSO) algorithm that dynamically adjusts the workload assigned to each processor, thereby making efficient use of all available processors in a heterogeneous cluster. The robustness, performance, and parallel efficiency of our PAPSO algorithm are compared to that of a parallel synchronous PSO (PSPSO) algorithm [5] using a suite of analytical test problems and a benchmark biomechanical test problem. Parallel performance of both algorithms is evaluated on homogeneous and heterogeneous Linux clusters using problems that require constant and variable amounts of computation time per function evaluation.

PARTICLE SWARM OPTIMIZATION

Synchronous vs asynchronous design

Particle swarm global optimization is a class of derivative-free, population-based computational methods introduced by Kennedy and Eberhart in 1995 [23]. Particles (design points) are distributed throughout the design space and their positions and velocities are modified based on knowledge of the best solution found thus far by each particle in the 'swarm'. Attraction towards the best-found solution occurs stochastically and uses dynamically-adjusted particle velocities. Particle positions (Equation (1)) and velocities (Equation (2)) are updated as shown below

$$x_{k+1}^{i} = x_{k}^{i} + v_{k+1}^{i} \tag{1}$$

$$v_{k+1}^{i} = w_{k} v_{k}^{i} + c_{1} r_{1} (p_{k}^{i} - x_{k}^{i}) + c_{2} r_{2} (p_{k}^{g} - x_{k}^{i}) \tag{2}$$

where x_k^i represents the current position of particle *i* in design space and subscript *k* indicates a (unit) pseudo-time increment. The point p_k^i is the best-found position of particle *i* up to time step *k* and represents the cognitive contribution to the search velocity v_k^i. The point p_k^g is the global best-found position among all particles in the swarm up to time step *k* and forms the social contribution to the velocity vector. The variable w_k is the particle inertia, which is reduced dynamically to decrease the search area in a gradual fashion (see References [2,5] for further details). Random numbers r_1 and r_2 are uniformly distributed in the interval [0, 1], while c_1 and c_2 are the cognitive and social scaling parameters, respectively (see References [5,24] for further details).

While several modifications to the original PSO algorithm have been made to increase robustness and computational throughput [24–28], one of the key issues is whether a synchronous or asynchronous approach is used to update particle positions and velocities. The sequential synchronous PSO algorithm updates all particle velocities and positions at the end of every optimization iteration (Figure 1). In contrast, the sequential asynchronous PSO algorithm updates particle positions and velocities continuously based on currently available information (Figure 2). The difference between the two methods is similar to the difference by Jacobi (synchronous) and Gauss–Seidel (asynchronous) methods for solving linear systems of equations. Carlisle and Dozier [29] evaluated the performance of both methods on analytical test problems using a sequential PSO algorithm that did not include dynamically-adjusted particle velocities. They concluded that the asynchronous approach is generally less costly computationally than is the synchronous approach based on the number of function evaluations required to achieve convergence. However, the performance of the two update methods were problem dependent and the differences were not significant.

Apart from one study published concurrently [30], only synchronous parallel implementations of a PSO algorithm have been developed thus far [5,6]. A synchronous approach maintains consistency between sequential and parallel implementations, thereby avoiding alteration of the convergence characteristics of the algorithm. Thus, parallel synchronous PSO should obtain exactly the same final solution as sequential synchronous PSO if the sequence of random numbers generated by the algorithm (i.e. r_1 and r_2 in Equation (2)) are constrained to be the same. Because the parallel performance of a synchronous implementation can suffer from load imbalance, PSPSO works best when there is no heterogeneity in either the computing environment or evaluation time for the analysis function and when the number of particles is an integer multiple of the number of processors.

Parallel asynchronous design

Current parallel computing environments and optimization tasks make it difficult to achieve the ideal conditions required for high parallel efficiency with PSPSO. To overcome this problem, we propose an adaptive concurrent strategy, called parallel asynchronous particle swarm optimization (PAPSO), which can increase parallel performance significantly. In addition to parallelization, this strategy involves one significant modification to the sequential asynchronous PSO algorithm. Since the updated velocity and position vectors of particle *i* are calculated using the best-found position p_k^i and the global best-found position p_k^g up to iteration *k*, the order of particle function evaluations affects the outcome of the optimization. However, the PSO algorithm does not restrict the order in which particle function evaluations are performed. Consequently, we permit the particle order to change continuously depending on the speed with which each processor completes its function evaluations, thereby eliminating the use of iteration counter k in our modified APSO algorithm.

The proposed PAPSO approach can be understood at a high level by comparing a block diagram of its algorithm with that of a PSPSO algorithm [5] (Figure 3). Our PAPSO algorithm updates particle positions and velocities continuously based on currently available information (Figure 3(a)). In contrast, PSPSO updates particle positions and velocities at the end of each optimizer iteration using global synchronization (Figure 3(b)). Thus, while PSPSO uses static load balancing to determine processor workload at compile time, PAPSO uses dynamic load balancing to determine processor workload during run time, thereby reducing load imbalance. The remainder of this section describes the key features incorporated into the proposed PAPSO algorithm.

Our PAPSO design follows a master/slave paradigm. The master processor holds the queue of particles ready to send to slave processors and performs all decision-making processes such as velocity/position updates and convergence checks. It does not perform any function evaluations. The slave processors repeatedly evaluate the analysis function using the particles assigned to them. The tasks performed by the master and slave processors are as follows:

- Master processor
  1. Initializes all optimization parameters and particle positions and velocities;
  2. Holds a queue of particles for slave processors to evaluate;
  3. Updates particle positions and velocities based on currently available information p^i, p^g;
  4. Sends the position x^i of the next particle in the queue to an available slave processor;
  5. Receives cost function values from slave processors;
  6. Checks convergence.
- Slave processor
  7. Receives a particle position from the master processor;
  8. Evaluates the analysis function $f(x^i)$ at the given particle position x^i;
  9. Sends a cost function value to the master processor.

Once the initialization step has been performed by the master processor, particles are sent to the slave processors to evaluate the analysis function. Consequently, the initial step of the optimization is identical to that of the PSPSO algorithm.

After initialization, the PAPSO algorithm uses a first-in-first-out centralized task queue to determine the order in which particles are sent to the slave processors (Figure 4). Whenever a slave processor completes a function evaluation, it returns the cost function value and corresponding particle number to the master processor, which places the particle number at the end of the task queue. Since the order varies in which particles report their results, randomness in the particle order occurs. Thus, unlike synchronous PSO, sequential and parallel implementations of asynchronous PSO will not produce identical results. Once a particle reaches the front of the task queue, the master processor updates its position and sends it to the next available slave processor. If the number of slave processors is the same as the number of particles, then the next available processor will always be the same processor that handled the particle initially. If the number of slave processors is less than the number of particles, then the next available processor will be whichever processor happens to be free when the particle reaches the front of the task queue. Even with heterogeneity in tasks and/or computational resources, the task queue ensures that each particle performs approximately the same number of function evaluations over the course of an optimization.

Communication between master and slave processors is achieved using a point-to-point communication scheme implemented with the Message Passing Interface [31,32]. Since there is no global synchronization in PAPSO, communication time is hidden within the computation time of the slave processors. Because the master processor can communicate with only one slave processor at a time, each slave processor remains idle for a short period of time while waiting to connect to the master processor after completing a function evaluation. However, this idle time is typically negligible compared to the computation time required for each function evaluation.

SAMPLE OPTIMIZATION PROBLEMS

Analytical test problems

To evaluate the performance of the proposed PAPSO algorithm, a suite of difficult analytical problems with known solutions [5,7,8,29] was used as test cases. Each problem in the suite was evaluated using the proposed PAPSO algorithm and a previously published PSPSO algorithm [5]. Each run was terminated based on a pre-defined number of function evaluations for the particular problem being solved. A detailed description of the four analytical test problems can be found in References [5,7,8,29]. The following is a brief description of the analytical test problems:

H1—This simple 2-dimensional function [7,8] has several local maxima and a global maximum of 2 at the co-ordinates (8.6998, 6.7665). Ten thousand function evaluations were used for this problem. An optimization was considered to be successful when the error between the true solution and the optimized solution was less than 0.001 (i.e. acceptable error) [8].

$$H_1(x_1,\ x_2) = \frac{\sin^2\left(x_1 - \frac{x_2}{8}\right) + \sin^2\left(x_2 + \frac{x_1}{8}\right)}{d+1}, \quad x_1,\ x_2 \in [-100,\ 100] \tag{3}$$

where $d = \sqrt{(x_1 - 8.6998)^2 + (x_2 - 6.7665)^2}$.

H2—This inverted version of the F6 function used by Schaffer *et al.* [33] has two design variables with several local maxima around the global maximum of 1.0 at (0, 0). This problem was solved using 20 000 function evaluations per optimization run, and the acceptable error was 0.001 [8].

$$H_2(x_1,\ x_2) = 0.5 - \frac{\sin^2\left(\sqrt{x_1^2 + x_2^2}\right) - 0.5}{(1 + 0.001(x_1^2 + x_2^2))^2}, \quad x_1,\ x_2 \in [-100,\ 100] \tag{4}$$

H3—This test function from Corona *et al.* [34] was used with dimensionality $n = 4$, 8, and 16. The function contains a large number of local minima (on the order of 10^{4n}) with a global minimum of 0 at $|x_i| < 0.05$. The number of function evaluations used for this problem was 50 000 ($n = 4$), 100 000 ($n = 8$), and 200 000 ($n = 16$). Acceptable error was 0.001 [8].

$$H_3(x_1, \dots, x_n) = \sum_{i=1}^{n} \begin{cases} (t \cdot \operatorname{sgn}(z_i) + z_i)^2 \cdot c \cdot d_i & \text{if } |x_i - z_i| < t \\ d_i \cdot x_i^2 & \text{otherwise} \end{cases} \tag{5}$$

$$x_i \in [-1000, 1000]$$

where $z_i = \left\lfloor \left| \frac{x_i}{s} \right| + 0.49999 \right\rfloor \cdot \operatorname{sgn}(x_i) \cdot s, \quad c = 0.15$

$$s = 0.2, \quad t = 0.05, \quad \text{and } d_i = \begin{cases} 1, & i = 1, 5, 9, \dots \\ 1000, & i = 2, 6, 10, \dots \\ 10, & i = 3, 7, 11, \dots \\ 100, & i = 4, 8, 12, \dots \end{cases}$$

H4—This test problem from Griewank [35] superimposes a high frequency sine wave on a multi-dimensional parabola. The function has a global minimum of 0 at $x_i = 0.0$, and the number of local minima increases exponentially with the number of design variables. To investigate large-scale optimization issues, problems were formulated using 32 and 64 design variables. The number of function evaluations used for this problem was 320 000 (n = 32) and 640 000 (n = 64) [5]. Acceptable error was 0.1 as recommended in Reference [29].

$$H_4(x_1, \dots, x_n) = \sum_{i=1}^{n} \frac{x_i^2}{d} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1, \quad x_i \in [-600, 600] \tag{6}$$

where d = 4000.

Biomechanical test problem

In addition to the analytical test problems, a real-life biomechanical test problem was included to evaluate PAPSO performance when different function evaluations require significantly different computation times, thereby creating a large load imbalance. The benchmark problem involves determination of patient-specific parameter values that permit a three-dimensional (3D) kinematic ankle joint model to reproduce experimental movement data as closely as possible (Figure 5) [4]. Twelve parameters (treated as design variables) specify the positions and orientations of joint axes fixed in adjacent body segments (i.e. shank, talus, and foot) within the 8 degree-of-freedom (DOF) kinematic ankle model. After specifying realistic values for the 12 joint parameters, we impose realistic motions on the joints in the model to create synthetic (i.e. computer-generated) trajectories for three surface markers fixed to the foot and lower leg. The goal of the optimization is to find the 12 joint parameters that allow the model to reproduce the motion of the six surface markers as closely as possible.

This system identification problem is solved via a two-level optimization approach. Given the current guess for the 12 parameters, a lower-level optimization (or sub-optimization) adjusts the DOFs in the model so as to minimize the 3D co-ordinate errors between modelled and synthetic marker locations for a single time frame. The upper-level optimization adjusts the 12 parameters defining the joint structure so as to minimize the sum of the cost function values calculated by the lower-level optimization over all time frames.

In mathematical form, the above system identification optimization problem can be stated as follows:

$$f(x) = \min_{p} \sum_{t=1}^{nf} e(p, t) \tag{7}$$

with

$$e(p,\ t) = \min_{q} \sum_{i=1}^{nm} \sum_{j=1}^{3} (a_{ij}(t) - b_{ij}(p,\ q))^2 \tag{8}$$

where Equation (7) is the cost function for the upper-level optimization. This equation is a function of patient-specific model parameters *p* and is evaluated for each of *nf* recorded time frames. Equation (8) represents the cost function for the lower-level optimization and uses a non-linear least-squares algorithm to adjust the model's DOFs (*q*) to minimize errors between experimentally measured marker co-ordinates a_{ij} and model-predicted marker co-ordinates b_{ij}, where *nm* is the number of markers whose three co-ordinates are used to calculate errors.

Evaluation metrics

A variety of metrics were used to quantify the performance, robustness, and parallel efficiency of the PAPSO algorithm compared to the PSPSO algorithm. For all PSO runs (asynchronous and synchronous), 20 particles were used unless otherwise noted, and PSO algorithm parameters were set to standard recommended values (see Reference [7] for details). Parallel efficiency was evaluated on two test beds. The first was a homogeneous Linux cluster of 20 identical machines connected with a Gigabit Ethernet switched network, where each machine possessed a 1.33 GHz Athlon processor and 256 MB of memory. The second was a group of 20 heterogeneous machines chosen from several Linux clusters (Table I). Parallel efficiency is defined as the ratio of speedup to the number of processors, where speedup is the ratio of sequential execution time to parallel execution time on a homogeneous cluster. Ideally, speedup should equal the number of processors and parallel efficiency should be 100%. However, parallel efficiency is normally less due to communication overhead, overhead caused by parallel decomposition of the optimization algorithm, and function evaluations that require different amounts of computation time. Evaluating the biomechanical test problem on a heterogeneous cluster introduced two sources of load imbalance into the evaluation process: the processors and the problem itself. Both test beds were located in the High-performance Computing and Simulation Research Laboratory at the University of Florida, and all tests were performed without other users on the selected processors.

For the analytical test problems, evaluation metrics were calculated from 100 optimization runs using different initial guesses. Robustness was measured as the fraction of runs that converged to within the specified tolerance of the known optimal cost function value. Performance was quantified as the mean and standard deviation of the number of function evaluations required to reach the optimal solution for runs that converged to the correct solution. Parallel efficiency was evaluated by running a modified version of one analytical test problem (H3 with $n = 16$) for 40 000 function evaluations on the homogeneous test bed. For this problem only, 32 particles were used with 1, 4, 8, 16, and 32 processors. Time delays of 0.5, 1, or 2 s were added to each function evaluation to increase the ratio of computation time to communication time. To mimic real-life variability, we increased the time delay for each function evaluation by up to 0, 20, or 50% of its nominal value using uniform random sampling. For example, a 2-second time delay randomly increased by up to 50% would produce delays ranging from 2 to 3 s (Table II).

For the biomechanical test problem, evaluation metrics were calculated from 10 optimization runs using 10 different initial guesses (PAPSO and PSPSO) and the same initial guess 10 times (PAPSO), thereby creating three cases for comparison. Due to the random nature of particle order in PAPSO, reusing the same initial guess will not produce the same results. Robustness was reported as the mean final cost function value and the associated root-mean-square (RMS) error in marker distance and joint parameter values. Performance was evaluated by plotting the average convergence history for each of the three cases. Parallel performance was quantified for both the homogeneous and the heterogeneous test bed. For the homogeneous test bed, execution time, speedup, and parallel efficiency were calculated for 1, 5, 10, and 20 processor systems. For the heterogeneous test bed, speedup and parallel efficiency cannot be calculated due to heterogeneity in the computational resources, so only a comparison of total execution time for the homogeneous vs heterogeneous test beds using 20 processors was reported.

RESULTS

Overall, PAPSO algorithm performance and robustness were comparable to that of the PSPSO algorithm for the analytical test problems. For 100 independent runs, both algorithms converged to the correct solution 100% of the time for problems H1, H3 with $n = 4$ and 8, and H4 with $n = 32$ (Table III). The fraction of unsuccessful runs for problems H2, H3 with $n = 16$, and H4 with $n = 32$ was approximately the same for both algorithms. Convergence speed for both algorithms was statistically the same ($p > 0.05$ based on Student's t-tests) on all problems except H2, where PSPSO performed slightly better (Table IV).

For the analytical test problem, PAPSO and PSPSO parallel efficiency exhibited different responses to increased number of processors, delay time, and delay time variation (Figure 6). PAPSO parallel efficiency was worse than that of PSPSO for small numbers of processors but generally better for large numbers of processors. While PSPSO parallel efficiency decreased approximately linearly with increasing number of processors to as low as 76% for 32 processors, PAPSO parallel efficiency increased non-linearly from only 75% for 4 processors to over 90% for 16 processors and 92% for 32 processors. These trends were sensitive to delay time and its variation for PSPSO but not PAPSO, with the sensitivity increasing with increasing number of processors. PSPSO parallel efficiency decreased when delay time was decreased from 2 to 0.5 s (i.e. decreased computation time) and when delay time variation was increased from 0 to 50%. For 32 processors, each 20% change in delay time variation decreased PSPSO parallel efficiency by about 5%.

For the biomechanical test problem, PAPSO again demonstrated comparable performance and robustness to that of PSPSO. After 10 000 function evaluations, final cost function values, RMS marker distance errors, RMS orientation parameter errors, and RMS position parameter errors were statistically the same ($p > 0.05$ based on pair-wise Student's t-tests) for PSPSO with 10 sets of initial guesses, PAPSO with 10 sets of initial guesses, and PAPSO with 1 set of initial guesses used 10 times (Table V). All three approaches successfully recovered ankle joint parameter values consistent with previously reported results [7]. Mean convergence history plots as a function of number of function evaluations also exhibited similar convergence speed for all three approaches (Figure 7).

Parallel performance on the biomechanical test problem was very different between the two PSO algorithms. For the homogeneous test bed, parallel performance showed similar trends to those found with the analytical test problem (Figure 8). Total execution time decreased and speedup increased for both algorithms as the number of processors increased. However, PAPSO execution time and speedup were worse (higher and lower, respectively) than that of PSPSO on the 5-processor system but better on the 10- and 20-processor systems (e.g. 7 h vs 10 h on the 20-processor system). In contrast, as the number of processors was increased from 5 to 20, parallel efficiency increased for PAPSO but decreased for PSPSO. Similar to execution time and speedup, parallel efficiency was worse for PAPSO than for PSPSO on the 5-processor system but better on the 10- and 20-processors systems. The parallel efficiency gap between the two algorithms increased with number of processors, reaching almost 30% for the 20-processor system. For the heterogeneous test bed with 20 processors, execution time was approximately 3.5 times less for PAPSO than for PAPSO (Figure 9). Compared to the homogeneous environment, execution time increased for both algorithms, but the increase was much smaller for PAPSO.

DISCUSSION

This study presented an asynchronous parallel algorithm for particle swarm optimization. The algorithm was evaluated using a set of analytical test problems and a biomechanical test problem involving system identification of a 3D kinematic ankle joint model. Speedup and parallel efficiency results were excellent when there was heterogeneity in computing resources or the optimization problem itself. For problems utilizing a large number of processors, small computation-to-communication time ratio, or large variability in computation time per function evaluation, the proposed PAPSO algorithm exhibits excellent parallel performance along with optimization performance and robustness comparable to that of PSPSO.

Parallel efficiency of the PAPSO algorithm exhibited an unusual trend as a function of the number of processors. For most parallel algorithms, parallel efficiency decreases as the number of processors increases because of increased overhead caused primarily by inter-processor communication. This observation explains why PSPSO parallel performance decreased with an increasing number of processors. However, PAPSO exhibited the opposite trend. For the 5-processor system, PSPSO used all five processors to evaluate the analysis function, whereas PAPSO used only four processors since the master processor does not perform function evaluations. Thus, the master processor was idle much of the time in the 5-processor system, reducing speedup and parallel efficiency. Since the negative impact of one idle processor decreases with increasing number of processors, PAPSO parallel performance improved rapidly as the number of processors was increased, asymptotically approaching 93%. In addition, communication overhead for the PAPSO algorithm was smaller than for the PSPSO algorithm, since PAPSO uses point-to-point communication between the master processor and each slave processor. A point-to-point communication scheme is very efficient when the computational cost varies for different function evaluations but less efficient when the computational cost is the same for all function evaluations.

The modified analytical test problem showed that both magnitude and variability of computation time for function evaluations affect parallel performance of PSPSO but not PAPSO. An increase in computation time through added constant time delay resulted in an increased computation-to-communication time ratio, improving parallel efficiency for PSPSO. In contrast, increasing the variability in computation time through time delay variations degraded PSPSO parallel efficiency due to load imbalance. Consequently, the parallel performance of PSPSO is relatively good when the computation-to-communication time ratio is high and when the computation time per function evaluation is relatively constant for any set of design variables used during the optimization. PSPSO may also exhibit poorer parallel performance when the parallel optimization involves a large number of processors due to increased communication overhead.

There are three primary limitations to the proposed PAPSO algorithm. First, parallel efficiency is poor for low numbers of processors, as discussed above. Second, two runs starting with the same particle locations in design space will not produce identical answers, though the same is true for PSPSO if the random number sequence generated by the algorithm is not fixed. Third, the degree of algorithm parallelism is limited by the number of particles, a problem shared with PSPSO. The maximum number of processors that can be used for PSPSO is equal to the number of particles and the number of particles plus one for PAPSO, putting an upper bound on performance improvements. However, PAPSO can achieve good parallel efficiency for any number of processors less than the maximum, whereas PSPSO can only achieve good parallel efficiency when the number of particles can be distributed in equal groups to the available processors. For both algorithms, if more processors are available than the number of particles, parallelization of the analysis function as well (i.e. parallelization on two levels) could be used to achieve improved scalability [13].

In summary, this study introduced a parallel asynchronous particle swarm optimization algorithm that exhibits good parallel performance for large numbers of processors as well as good optimization robustness and performance. Since PAPSO incorporates a dynamic load balancing scheme, parallel performance is dramatically increased for (1) heterogeneous computing environments, (2) user-loaded computing environments, and (3) problems producing run-time load variations. For a sample problem where the computational cost of the analysis function varied for different sets of design variables, PAPSO achieved better than 90% parallel efficiency on a homogeneous 20-processor system while PSPSO achieved only 62%. For the same problem run on a heterogeneous 20-processor system, PAPSO finished a given number of function evaluations 3.5 times faster than did PSPSO. However, PAPSO performance is worse than that of PSPSO for small numbers of processors and/or if the time for each function evaluation is large (compared to communication time) and constant throughout the optimization. Overall, PAPSO provides a valuable option for parallel global optimization under heterogeneous computing conditions.

Acknowledgements

This study was funded by NIH National Library of Medicine (R03 LM07332) and Whitaker Foundation grants to B.J. Fregly. We thank Jaco Schutte for providing the parallel synchronous PSO algorithm used for the evaluations.

References

1. Venter, G.; Sobieszczanski-Sobieski, J. 9th AI AA/I SSMO Symposium on Multidisciplinary Analysis and Optimization. Atlanta, GA: 2002. Multidisciplinary optimization of a transport aircraft wing using particle swarm optimization.
2. Fourie, PC.; Groenwold, AA. The particle swarm algorithm in topology optimization; Proceedings of the 4th World Congress of Structural and Multidisciplinary Optimization; Dalian, China. May 2001;
3. Eberhart, RC.; Shi, Y. Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2001). IEEE Press; New York: Particle swarm optimization: developments, applications and resources; p. 81-86.
4. Reinbolt JA, Schutte JF, Fregly BJ, Koh BI, Haftka RT, George AD, Mitchell KH. Determination of patient-specific multi-joint kinematic models through two-level optimization. Journal of Biomechanics 2005;38:621–626. [PubMed: 15652563]
5. Schutte JF, Reinbolt JA, Fregly BJ, Haftka RT, George AD. Parallel global optimization with particle swarm algorithm. International Journal for Numerical Methods in Engineering 2004;61:2296–2315. [PubMed: 17891226]
6. Gies D, Rahmat-Smil Y. Particle swarm optimization for reconfigurable phase-differentiated array design. Microwave and Optical Technology Letters 2003;38:168–175.
7. Schutte JF, Koh BI, Reinbolt JA, Fregly BJ, Haftka RT, George AD. Evaluation of a particle swarm algorithm for biomechanical optimization. Journal of Biomechanical Engineering 2005;127:465–474. [PubMed: 16060353]
8. van Soest JK, Casius LJR. The merits of a parallel genetic algorithm in solving hard optimization problems. Journal of Biomechanical Engineering 2003;125:141–146. [PubMed: 12661208]
9. Pereiraa CMNA, Lapaa CMF. Coarse-grained parallel genetic algorithm applied to a nuclear reactor core design optimization problem. Annals of Nuclear Energy 2003;30:555–565.
10. Higginson JS, Neptune RR, Anderson FC. Simulated parallel annealing within a neighborhood for optimization of biomechanical systems. Journal of Biomechanics 2004;38:1938–1942. [PubMed: 16023483]
11. Hong CE, McMillin BM. Relaxing synchronization in distributed simulated annealing. IEEE Transactions on Parallel and Distributed Systems 1995;6(2):189–195.

12. White DNJ. Parallel pattern search energy minimization. Journal of Molecular Graphics and Modelling 1997;15(3):154–157. [PubMed: 9457617]
13. Koh BI, Reinbolt JA, Fregly BJ, George AD. Evaluation of parallel decomposition methods for biomechanical optimizations. Computer Methods in Biomechanics and Biomedical Engineering 2004;7:215–225. [PubMed: 15512765]
14. Pandy MG. Computer modeling and simulation of human movement. Annual Reviews of Biomedical Engineering 2001;3:245–273.
15. Anderson FC, Pandy MG. A Dynamic optimization solution for vertical jumping in three dimensions. Computer Methods in Biomechanics and Biomedical Engineering 1992;2:201–231. [PubMed: 11264828]
16. Anderson FC, Ziegler JM, Pandy MG. Application of high-performance computing to numerical simulation of human movement. Journal of Biomechanical Engineering 1995;117:155–157. [PubMed: 7609481]
17. Conforti D, Musmanno R. Convergence and numerical results for a parallel asynchronous quasi-Newton method. Journal of Optimization Theory and Applications 1995;84:293–310.
18. Fischer H, Ritter K. An asynchronous parallel Newton method. Mathematical Programming 1988;42:363–374.
19. Hough PD, Kolda TG, Torcjon VJ. Asynchronous parallel pattern search for nonlinear optimization. SIAM Journal on Scientific Computing 2001;23:134–156.
20. Hart, WE.; Baden, S.; Belew, RK.; Kohn, S. Analysis of the numerical effects of parallelism on a parallel genetic algorithm; Proceedings of the 10th International Parallel Processing Symposium (IPPS ’96); Honolulu, HI. 15–19 April, 1996;
21. Alba E, Troya JM. Analyzing synchronous and asynchronous parallel distributed genetic algorithms. Future Generation Computer Systems 2001;17(4):451–465.
22. Lee S, Lee KG. Synchronous and asynchronous parallel simulated annealing with multiple Markov chains. IEEE Transactions on Parallel and Distributed Systems 1996;7(10):993–1008.
23. Kennedy, J.; Eberhart, RC. Particle swarm optimization; Proceedings of IEEE International Conference on Neural Networks; Perth, Australia. 1995; 1942–1948.
24. Shi, Y.; Eberhart, RC. Parameter selection in particle swarm optimization. In: Porto, VW.; Saravanan, N.; Waagen, D.; Eiben, AE., editors. Evolutionary Programming VII. Lecture Notes in Computer Science. 1447. Springer; Berlin: 1998. p. 591-600.
25. Shi, Y.; Eberhart, RC. A modified particle swarm optimizer; Proceedings of IEEE International Conference on Evolutionary Computation; Anchorage, Alaska. 1998. p. 69-73.
26. Eberhart, RC.; Shi, Y. Proceedings of 2000 Congress on Evolutionary Computation. Piscataway, NJ: IEEE Service Center; 2000. Comparing inertia weights and constriction factors in particle swarm optimization; p. 84-88.
27. Clerc, M. The swarm and the queen: Towards a deterministic and adaptive particle swarm optimization. In: Angeline, PJ.; Michalewicz, Z.; Schoenauer, M.; Yao, X.; Zalzala, A., editors. Proceedings of the Congress of Evolutionary Computation. 3. IEEE Press; Washington DC, U.S.A.: 1999. p. 1951-1957.
28. Løvbjerg, M.; Rasmussen, TK.; Krink, T. Hybrid particle swarm optimizer with breeding and subpopulations; Proceedings of the 3rd Genetic and Evolutionary Computation Conference (GECCO-2001); 2001;
29. Carlisle, A.; Dozier, G. Proceedings of the Workshop on Particle Swarm Optimization. Indianapolis, U.S.A.: 2001. An off-the-shelf PSO.
30. Venter G, Sobieszczanki-Sobieksi J. A parallel particle swarm optimization algorithm accelerated by asynchronous evaluations. Journal of Aerospace Computing, Information, and Communication. in press
31. Snir, M.; Otto, S.; Huss-Lederman, S.; Walker, D.; Dongarra, J. MPI: The Complete Reference. Cambridge: MIT Press; 1996.
32. Message Passing Interface Forum, MPI. Technical Report CS-94-230. Computer Science Department, University of Tennessee; 1 April. 1994 a message-passing interface standard.
33. Schaffer, JD.; Caruana, RA.; Eshelman, LJ.; Das, R. A study of control parameters affecting online performance of genetic algorithms for function optimizing. In: David, JD., editor. Proceedings of the

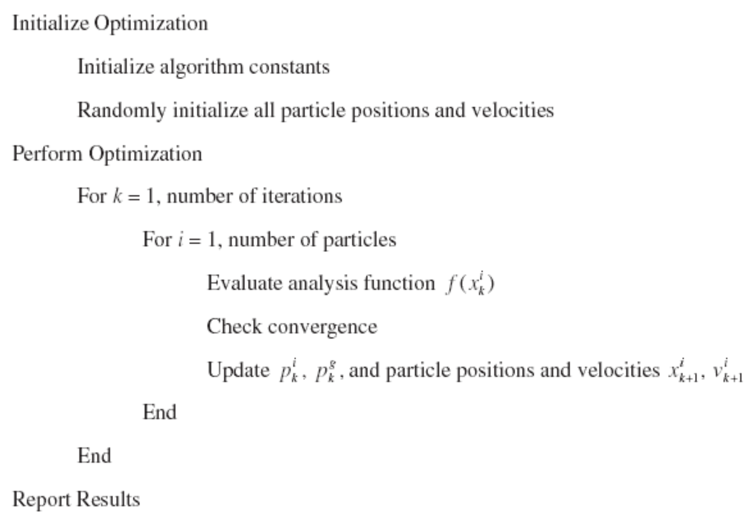
3rd International Conference on Genetic Algorithm. Morgan Kaufmann Publishers; San Mateo, CA: 1989. p. 51-60.

34. 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.

35. Griewank AO. Generalized descent for global optimization. Journal of Optimization Theory and Applications 1981;34:11–39.

```
Initialize Optimization
    Initialize algorithm constants
    Randomly initialize all particle positions and velocities
Perform Optimization
    For $k$ = 1, number of iterations
        For $i$ = 1, number of particles
            Evaluate analysis function $f(x_k^i)$
        End
        Check convergence
        Update $p_k^i$, $p_k^g$, and particle positions and velocities $x_{k+1}^i$, $v_{k+1}^i$
    End
Report Results
```

Figure 1.
Pseudo-code for a sequential synchronous PSO algorithm.



```
Initialize Optimization
    Initialize algorithm constants
    Randomly initialize all particle positions and velocities
Perform Optimization
    For $k$ = 1, number of iterations
        For $i$ = 1, number of particles
            Evaluate analysis function $f(x_k^i)$
            Check convergence
            Update $p_k^i$, $p_k^g$, and particle positions and velocities $x_{k+1}^i$, $v_{k+1}^i$
        End
    End
Report Results
```

Figure 2.
Pseudo-code for a sequential asynchronous PSO algorithm.

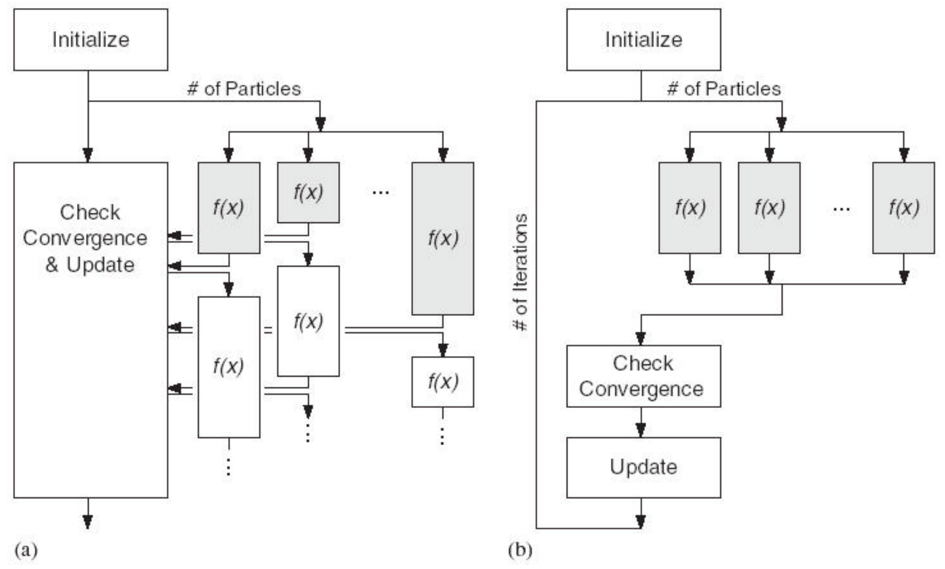


Figure 3.
Block diagrams for: (a) parallel asynchronous; and (b) parallel synchronous PSO algorithms. Grey boxes indicate first set of particles evaluated by each algorithm.

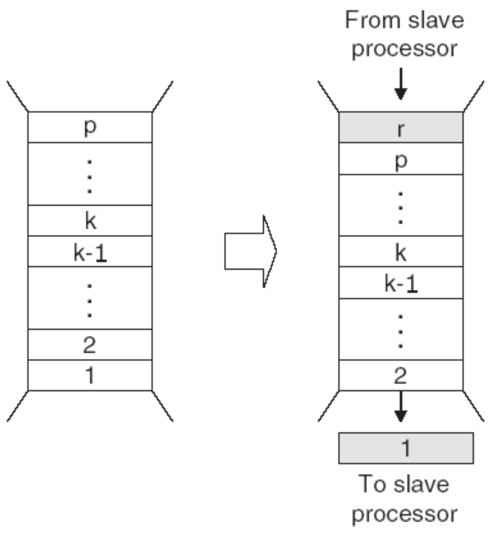


Figure 4.
Block diagram for first-in-first-out centralized task queue with *p* particles on a *k*-processor system. After receiving the cost function value and corresponding particle number *r* from a slave processor, the master processor stores the particle number at the end of the queue, updates the position of the first particle, and sends the first particle back to idle slave processor for evaluation. The particle order changes depending on the speed with which each processor completes its function evaluations.

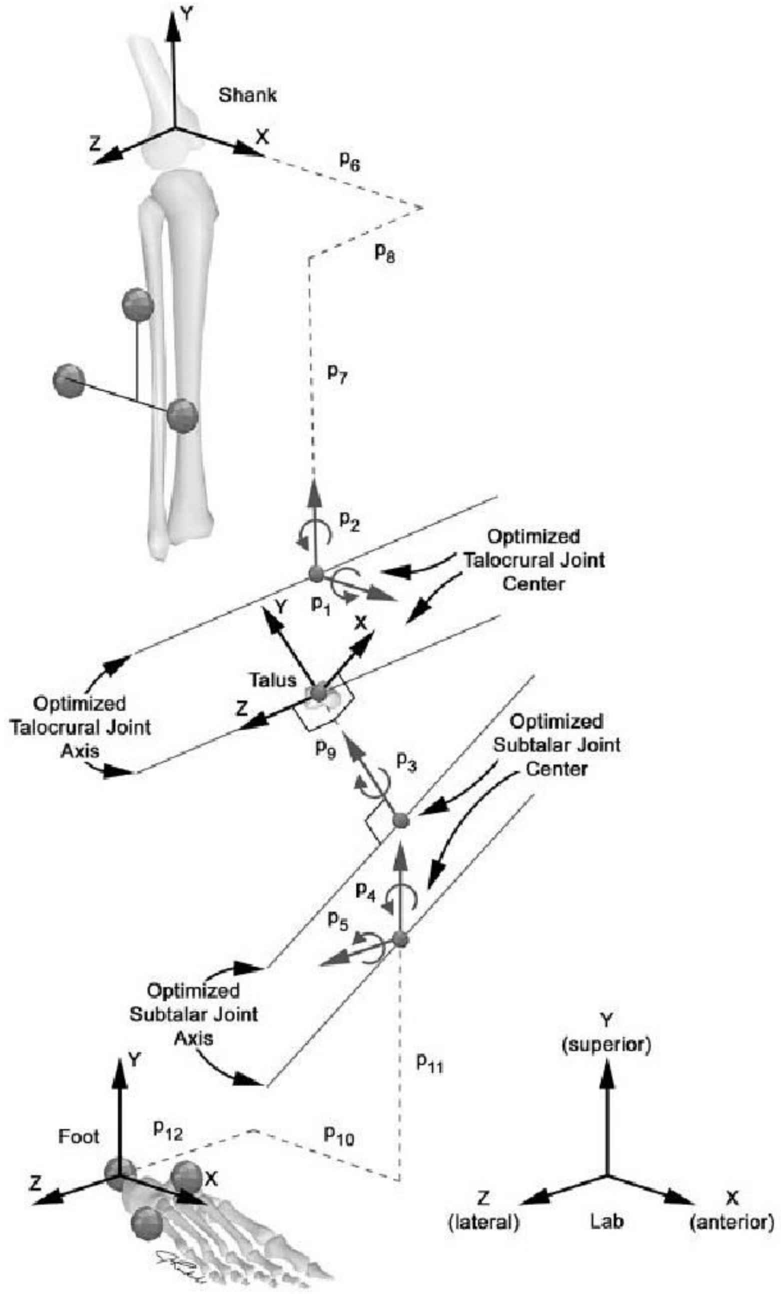


Figure 5.
Kinematic ankle joint model used in the biomechanical test problem. The model is three-dimensional, possesses eight degrees of freedom (six for the position and orientation of the shank segment and two for the ankle joint), and requires 12 parameters p_1–p_{12} to define the positions and orientations of the joint axes fixed in the shank, talus, and foot segment co-ordinate systems.

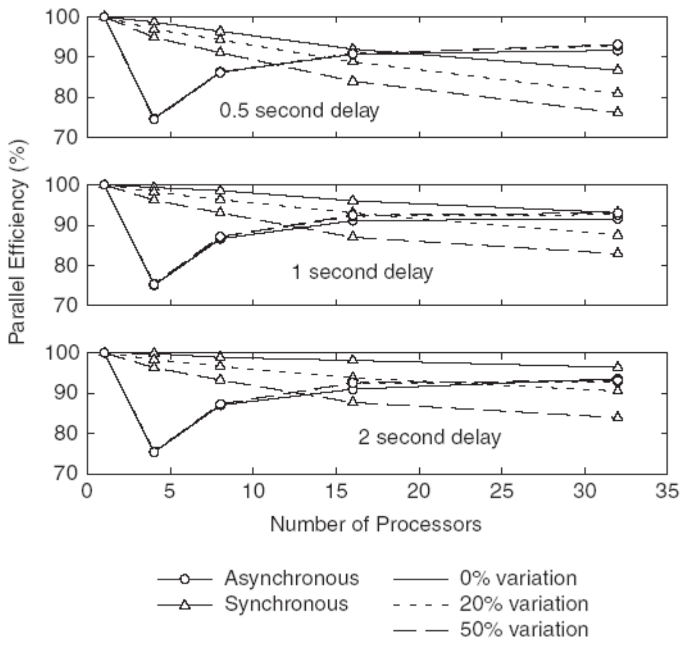


Figure 6.
Parallel efficiency for the parallel asynchronous (circles) and parallel synchronous (triangles) PSO algorithms as a function of number of processors, computational delay time (0.5, 1, or 2 s), and computational delay time variation (0, 20, or 50%). The optimization used 32 particles for analytical test problem H3 with *n* = 16 and was performed for 40 000 function evaluations on a homogeneous cluster.

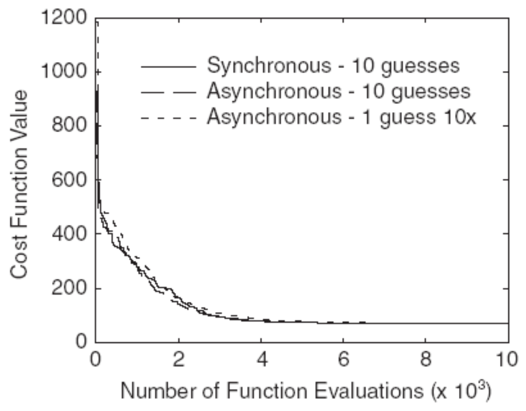


Figure 7.
Mean convergence history plots over 10 000 function evaluations for the biomechanical test problem obtained with the parallel asynchronous and parallel synchronous PSO algorithms. Each set of 10 optimizations used either 10 different sets of initial guesses (synchronous and asynchronous) or 1 set of initial guesses 10 times (asynchronous). Refer to Table V for corresponding quantitative results.

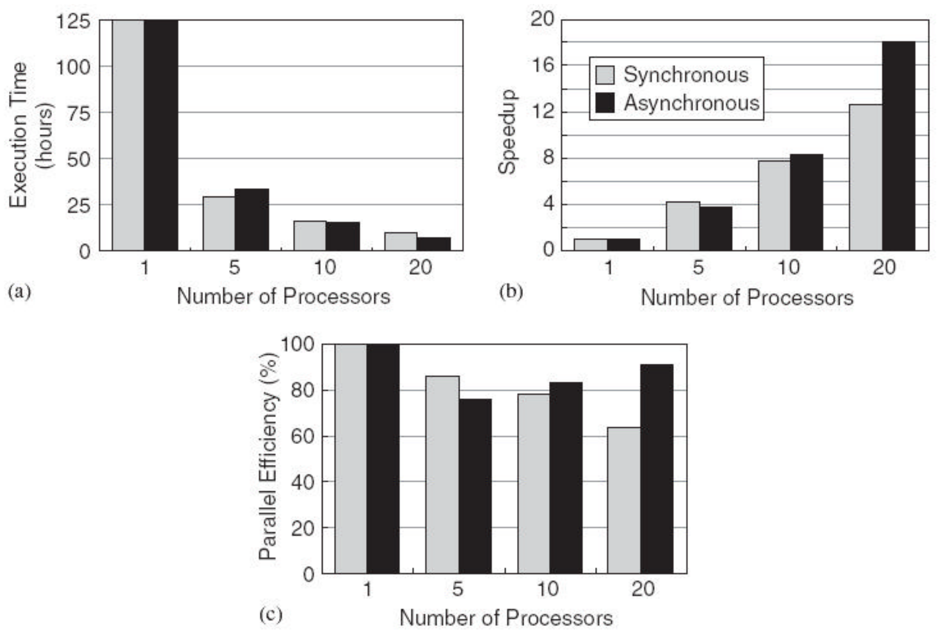


Figure 8.
Execution time (a), speedup (b), and parallel efficiency (c) for parallel asynchronous and parallel synchronous PSO algorithms for the biomechanical test problem in a homogeneous computing environment.

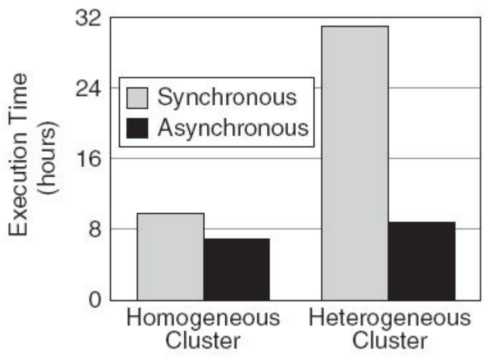


Figure 9.
Execution time for parallel asynchronous and parallel synchronous PSO algorithms for the biomechanical test problem in a heterogeneous computing environment.

Table I

Summary of heterogeneous computing resources used to evaluate execution time for the biomechanical test problem.

CPU speed	CPU type	Memory	Network	Nodes
2.4 GHz	Intel Xeon	1 GB	Gigabit Ethernet	3
1.4 GHz	AMD Opteron	1 GB	Gigabit Ethernet	3
1.3 GHz	AMD Athlon	256 MB	Gigabit Ethernet	3
1.0 GHz	Intel Pentium-III	256 MB	Fast Ethernet	3
733 MHz	Intel Pentium-III	256 MB	Fast Ethernet	3
600 MHz	Intel Pentium-III	256 MB	Fast Ethernet	3
400 MHz	Intel Pentium-II	128 MB	Fast Ethernet	2

Table II

Range of computation time delays [min, max] for each function evaluation of problem H3 with $n = 16$. Times are uniformly distributed within the indicated interval.

	Delay time (s)		
Variation (%)	**0.5**	**1**	**2**
0	[0.5, 0.5]	[1.0, 1.0]	[2.0, 2.0]
20	[0.5, 0.6]	[1.0, 1.2]	[2.0, 2.4]
50	[0.5, 0.75]	[1.0, 1.5]	[2.0, 3.0]

Table III

Robustness results for analytical test problems. Values are fraction of 100 optimization runs that successfully reached the correct solution to within the specified tolerance (see text). *n* indicates the number of design variables used for each problem.

	Analytical test problem						
	H1	**H2**	**H3**			**H4**	
Parallel PSO algorithm	**(*n* = 2)**	**(*n* = 2)**	**(*n* = 4)**	**(*n* = 8)**	**(*n* = 16)**	**(*n* = 32)**	**(*n* = 64)**
Synchronous	1.00	0.61	1.00	1.00	0.82	1.00	0.96
Asynchronous	1.00	0.61	1.00	1.00	0.82	1.00	0.91

Table IV

Performance results for analytical test problems. Values are mean (standard deviation) number of function evaluations required to reach the final solution for optimization runs that converged to the correct solution.

	Analytical test problem						
	H1	**H2**	**H3**			**H4**	
Parallel PSO algorithm	**(*n* = 2)**	**(*n* = 2)**	**(*n* = 4)**	**(*n* = 8)**	**(*n* = 16)**	**(*n* = 32)**	**(*n* = 64)**
Synchronous	3396 (376)	6243 (4280)	4687 (452)	7549 (701)	24 918 (5108)	15 403 (1294)	29 843 (2790)
Asynchronous	3495 (385)	6594 (4171)	4560 (444)	7577 (576)	25 036 (6330)	15 427 (1449)	29 949 (2553)

Table V

Mean final cost function values and associated RMS marker distance and joint parameter errors after 10 000 function evaluations obtained with the parallel synchronous and asynchronous PSO algorithms. Standard deviations are indicated in parentheses. Each set of 10 optimizations used either 10 different sets of initial guesses (synchronous and asynchronous) or 1 set of initial guesses 10 times (asynchronous).

			RMS Error		
Parallel PSO algorithm	**Initial guesses**	**Cost function value**	**Marker distances (mm)**	**Orientation parameters (deg)**	**Position parameters (mm)**
Synchronous	10	70.40 (1.18)	5.49 (0.04)	4.85 (3.00)	2.40 (1.44)
Asynchronous	10	69.92 (0.84)	5.47 (0.03)	3.19 (2.28)	2.39 (1.45)
Asynchronous	1	70.65 (1.01)	5.50 (0.04)	4.89 (2.47)	2.51 (1.34)