

PARALLEL ASYNCHRONOUS PARTICLE SWARM FOR GLOBAL BIOMECHANICAL OPTIMIZATION

¹Byung-Il Koh, ^{2,3}Benjamin J. Fregly, ¹Alan D. George, and ²Raphael T. Haftka

¹Department of Electrical & Computer Engineering, University of Florida, Gainesville, FL
²Department of Mechanical & Aerospace Engineering, University of Florida, Gainesville, FL

³Department of Biomedical Engineering, University of Florida, Gainesville, FL

Email: fregly@ufl.edu; Web: www.mae.ufl.edu/~fregly

INTRODUCTION

Parallel optimization uses multiple computers or processors in tandem to reduce the elapsed time required to solve optimization problems. Genetic [1], particle swarm [2], and simulated annealing [3] algorithms have recently been implemented in parallel to solve complex biomechanical optimization problems. All of these implementations are synchronous, meaning that an optimizer iteration cannot be completed until all function evaluations in the current iteration have been performed. Synchronous optimization works best when two 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 requires a constant evaluation time for any set of design variables throughout the optimization. If these two conditions are not met, the parallel optimization algorithm will not make the most efficient use of the available processors.

In this paper, we propose 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 accuracy, robustness, and parallel performance of the PAPSO algorithm are demonstrated using a biomechanical system identification problem for a three-dimensional (3D) kinematic ankle joint model [4].

METHODS

Particle Swarm Optimization (PSO): Particle swarm global optimization is a class of derivative free, population-based computational methods. In PSO, particles (design points) are distributed throughout the design space and their positions and velocities modified based on knowledge of the best solution found thus far by each particle in the “swarm.” Attraction toward the best-found solution occurs stochastically and uses dynamically adjusted particle velocities. Updating of particle positions (Eq. 1) and velocities (Eq. 2) are performed as shown below:

$$x_{k+1}^i = x_k^i + v_{k+1}^i \quad (1)$$

$$v_{k+1}^i = wv_k^i + c_1r_1(p_k^i - x_k^i) + c_2r_2(p_k^g - x_k^i) \quad (2)$$

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. 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 [2] for further details).

Parallel Asynchronous Particle Swarm Optimization: The PAPSO algorithm is derived from the parallel synchronous PSO (PSPSO) algorithm proposed by Schutte et al. [2]. The main difference between synchronous and asynchronous PSO implementations is the method used to update particle positions and velocities. Since PSPSO updates particle positions and velocities at the end of every optimizer iteration using global synchronization, it uses static load balancing where the workload assigned to each processor is determined at compile-time. In contrast, PAPSO updates particle positions and velocities continuously based on currently available information. Consequently, PAPSO incorporates a dynamic load balancing scheme with a centralized task queue approach to reduce load imbalance.

Our PAPSO implementation follows a master/slave model. The master processor holds the queue of particles ready to send to the slave processors. Whenever a slave processor completes a function evaluation, it sends the fitness value to the master processor. The master receives the fitness value, checks for convergence, and updates the particle’s position and velocity. The updated particle position is sent to the slave processor to perform another function evaluation.

Description of Analysis Function: A 3D biomechanical system identification problem [4] was used to evaluate our PAPSO algorithm. The problem involves determination of patient-specific parameter values that permit a 3D kinematic ankle joint model to reproduce experimental movement data as closely as possible. Twelve parameters (treated as design variables) specify the fixed positions and orientations of joint axes in adjacent body segments (i.e., shank, talus, and foot) within the 8 degree-of-freedom (DOF) kinematic ankle model.

The system identification problem is solved via a two-level optimization approach. Given the current guess for the 12 parameters, the lower-level optimization (or sub-optimization) adjusts DOFs in the model so as to minimize the 3D coordinate errors between modeled and experimental surface markers. The upper-level optimization adjusts the 12 parameters defining the joint structure so as to minimize the cost function calculated by the lower-level optimization over all time frames. Since the upper-level optimization includes the lower-level optimization, the time for each function evaluation can vary depending on the evaluation time required by the repeated lower-level optimizations.

RESULTS AND DISCUSSION

To ensure that accuracy and robustness were not degraded by the asynchronous implementation, 10 optimization runs with 10 different initial guesses were performed using both the PAPSO and PSPSO algorithms (Table 1). In addition, 10 runs

with the same initial guess were performed with PAPS0 since the algorithm can produce different results for the same initial guess. The results show that the accuracy and robustness of PAPS0 is comparable to that of PSPSO for this biomechanical optimization problem.

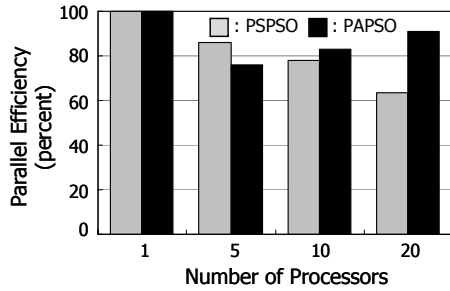


Figure 1: Parallel efficiency for PSPSO and PAPS0 on homogeneous processors as a function of the number of processors (PC cluster with 1.3GHz processor, 256MB RAM, Gigabit Ethernet network).

For parallel performance, we tested PAPS0 and PSPSO on two different test beds (Linux-based PC clusters in the University of Florida HCS Research Laboratory): one homogeneous (i.e., all processors the same) with 1, 5, 10, and 20 processors, and the other heterogeneous with 20 processors. For the homogeneous environment, parallel efficiency (i.e., the percentage of available processor time actually used) was generally higher for PAPS0 than for PSPSO, especially as the number of processors increased (Fig. 1, 91% vs. 64% parallel efficiency and 7 hours vs. 10 hours elapsed time in the 20-processor system). The one exception was for the lowest number of processors, where PSPSO was more efficient. In the 5-processor system, PAPS0 only used 4 processors to evaluate the analysis function, since the master processor was used to coordinate the particle queue and to send new particle positions to available slave processors. Thus, the master processor was idle most of the time in the 5-processor system, decreasing the parallel efficiency.

For the heterogeneous environment, parallel efficiency cannot be calculated, but execution time increased for both the PAPS0 and PSPSO algorithms compared to the homogeneous environment (Fig. 2). However, the effect of changing the computing environment was low on PAPS0 and high on PSPSO, with PAPS0 being approximately 3.5 times faster than PSPSO.

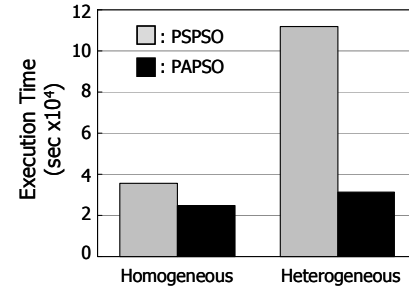


Figure 2: Execution time for PSPSO and PAPS0 on a heterogeneous 20-processor system (PC cluster utilizing two 400MHz/128MB processors with Fast Ethernet Network and three 600MHz/256MB, 733MHz/256MB, 1.0GHz/256MB, 1.3GHz/256MB, 1.4 GHz/1GB, and 2.4GHz/1GB processors with Gigabit Ethernet network).

CONCLUSIONS

This study presented an asynchronous parallel algorithm for particle swarm optimization. The algorithm was applied to a biomechanical optimization problem involving system identification of a 3D kinematic ankle joint model. Since PAPS0 incorporates a dynamic load balancing scheme, parallel performance is dramatically increased with respect to (1) heterogeneous computing environments, (2) user-loaded computing environments, and (3) run-time load variations. However, performance improvements are poorer for small number of processors and/or if the time for each function evaluation is constant throughout the optimization. An interesting direction for future research would be to apply fault-tolerant strategies for long-running parallel optimization algorithms to increase their robustness.

REFERENCES

1. van Soest JK and Casius LJR. *J Biomech Eng* **125**, 141-146, 2003.
2. Schutte JF et al. *Intl J Numerical Methods Eng* **61**, 2296-2315, 2004.
3. Higginson JS et al. *J. Biomech* (in press).
4. Reinbolt JA et al. *J Biomech* **38**, 621-626, 2005.

ACKNOWLEDGMENTS

This study was funded by NIH National Library of Medicine grant R03 LM07332 to B.J.F. and AFOSR grant F49620-09-1-0070 to R.T.H.

Table 1: Mean final cost function values and associated marker distance and joint parameter RMS errors after 10,000 function evaluations performed by PSPSO and PAPS0

Parallel algorithms	# of initial guess	Cost Function	RMS Error		
			Marker Distances (mm)	Orientation Parameters (deg)	Position Parameters (mm)
PSPSO	10	70.40	5.49	4.85	2.40
PAPS0	10	69.52	5.47	3.19	2.39
	1	70.65	5.50	4.89	2.51