A New Multi-Swarm Particle Swarm Optimization and Its Application to Lennard-Jones Problem

KUSUM DEEP[1]
MADHURI ARYA[2]
SHASHI BARAK[2]

Department of Mathematics,
Indian Institute of Technology Roorkee,
Roorkee-247667, Uttarakhand, India
[1]kusumfma@iitr.ernet.in
[2](madhuriiitr,shashibarak)@gmail.com

Abstract. Particle swarm optimization (PSO) algorithm is a modern heuristic technique for global optimization. Due to its ease of implementation, excellent effectiveness, and few parameters to adjust it has gained a lot of attention in the recent years. However, with the increasing size and computational complexity of real life optimization problems it takes long solution times and the solution quality also degrades, so there is a constant need to improve its effectiveness and robustness to find better solution in the shortest possible computational time. Parallel computing is a possible way to fulfill this requirement. In this paper we propose a multi-swarm approach to parallelize PSO algorithm (MSPSO). The performance of the proposed algorithm is evaluated using several well-known numerical test problems taken from literature. Then, it is applied to the challenging problem of finding the minimum energy configuration of a cluster of identical atoms interacting through the Lennard-Jones potential. Finding the global minimum of this function is very difficult because of the presence of a large number of local minima, which grows exponentially with molecule size. Computational results for clusters containing 8 and 9 atoms are obtained. The parallel algorithm shows significant speed-up without compromising the accuracy, when compared to the sequential PSO.

Keywords: Lennard-Jones potential, Parallel Particle swarm optimization, Parallel Computing.

(Received March 04, 2010 / Accepted September 15, 2010)

1 Introduction

Particle Swarm optimization (PSO) is an efficient and powerful population-based stochastic search technique for solving global optimization problems, which has been widely applied in many scientific and engineering fields. It is based on the social behavior metaphor [8]. It is sometimes regarded as an alternative tool to genetic algorithms (GAs) and other evolutionary algorithms (EAs). Due to its robustness, efficiency and simplicity PSO is becoming popular day by day. But just like other population based meta-heuristics PSO suffers from two difficulties: (i) falling into local optima for large problem instances of some hard problems, (ii) long execution times for time consuming objective functions. Parallel computing can be used to overcome these difficulties i.e., to accelerate the solution process and to improve the exploration capability of PSO. Also PSO algorithms are population based and so they are naturally suited to parallel implementation. These observations lead us to consider a parallel version of the Particle Swarm optimization algorithm. However, the literature on parallel meta-heuristics shows that finding an efficient parallel implementation of a sequential meta-heuristic is not an easy task.

With the growing popularity of PSO the researches concerning its parallelization are also increasing. Generally meta-heuristics are parallelized for two main reasons: (i) given a fixed time to search, the aim is to increase the quality of the solutions found in that time; (ii) given a fixed solution quality, the aim is to reduce the time needed to find a solution not worse than that quality. In this paper we aim to achieve the second goal. The strategies used in literature for the parallelization of PSO can be divided into two main categories. First are the master-slave parallelization strategies in which a processor (the master) distributes particles of a single swarm to many processors (slaves) for concurrent fitness evaluation. Master-slave approach is most suitable whenever the fitness evaluations are significantly computationally expensive. The second are the multi-swarm parallelization strategies which aim to partition the entire swarm into several sub swarms that are assigned to different processors and use some kind of information sharing among sub swarms.

A parallel version of PSO was first implemented by Schutte et al. [13]. They used a coarse grained master-slave parallelization approach. The implementation was based on a synchronous scheme. They reported excellent parallel performance for problems in which the individual fitness evaluations require same amount of time. They concluded that for problems where the time required for each fitness evaluation varies substantially, an asynchronous implementation may be needed to maintain high parallel efficiency. The synchronous approach leads to poor parallel speedup in cases where a heterogeneous parallel environment is used and/or where the analysis time depends on the design point being analyzed. Motivated by this Venter and Sobieski [15] introduced a parallel asynchronous PSO (PAPSO) and compared it with a parallel synchronous PSO (PSPSO). They reported that PAPSO significantly outperforms PSPSO in terms of parallel efficiency and gives numerical accuracy comparable to that of PSPSO.

Koh et al. [11] also proposed a PAPSO and compared it with PSPSO in homogeneous and heterogeneous computing environments. They reported that PAPSO gives excellent parallel performance when a large number of processors is used and either (1) heterogeneity exists in the computational task or environment, or (2) the computation-to-communication time ratio is relatively small. Chu and Pan [4] presented a parallel version of PSO (PPSO) together with three communication strategies that can be used according to independence of the data. They showed the usefulness of PPSO with their proposed communication strategies. Sahin et al. [12] implemented the PPSO algorithm on a computer cluster and studied the performance of the distributed PSO algorithm. Kim et al. [10] applied a parallel version of PSO based on coarse grained model for the solution of optimal power flow problem. In their model each subpopulation exchanges require information only between the adjacent subpopulations. With each processor that could communicate with the neighboring sub-populations, the best solution of each processor was transferred to the neighboring processors. Waintraub et al. [16] proposed three PPSO models inspired by traditional parallel GA models and applied them to two complex and time consuming nuclear engineering problems. The master-slave, island (ring topology) and cellular PPSO models were adapted from PGA. They proposed another PPSO based on a different approach. The idea was to use the concept of neighborhood in PSO, to connect islands, avoiding necessity of defining the "migration interval" parameter. They called it Neighborhood-Island PPSO. In the master-slave PPSO, the original PSO algorithm is not modified and no improvement in solutions is obtained. They found outstanding gains in terms of speedup of the optimization processes, due to parallelism that occur in all models. Due to the reduced amount of communication needed, island (with periodic migration) and Neighborhood-Island models were demonstrated to be fastest in all problems considered in their work.

The most common observation in the above-mentioned real-world applications is the focus on speedup due to parallel processing, which is also the aim of our proposed approach. Our approach is new and different from the approaches existing in the literature as it does not use the information sharing strategy among the swarms until some particular stage and after that stage too it uses a different idea as described in the section 3 of this paper. We present an implementation of our parallelization scheme based on MPI (Message Passing Interface) [14]. MSPSO is simulated to test its efficiency by solving some benchmark problems and the results are analyzed. A real life problem namely Lennard-John (L-J) problem is then considered to test the robustness of the proposed method. As far as we are aware this approach has never been used so far for the solution of L-J problem.

2 Basic PSO

PSO is a relatively newer addition to the class of population based search techniques. The concept of PSO was first suggested by Kennedy and Eberhart. Since its development in 1995, PSO has become one of the most promising optimization techniques for solving global optimization problems. Social insects (e.g. birds, fish,

ants etc.) have high swarm intelligence [9]. The mechanism of PSO is inspired by social and cooperative behavior displayed by various species like birds, fish etc. The PSO system consist of a population (swarm) of potential solutions called particles. Each particle has an associated fitness value. These particles move through search space with a specified velocity in search of optimal solution. Each particle maintains a memory which helps it in keeping the track of the best position it has achieved so far. This is called the particle's personal best position (pbest) and the best position the swarm has achieved so far is called global best position (gbest). PSO has two primary operators: Velocity update and Position update. During each generation each particle is accelerated towards the gbest and its own pbest. At each iteration a new velocity value for each particle is calculated according to the following velocity update equation:

$$v_{id} = wv_{id} + c_1 r_1(p_{id} - x_{id}) + c_2 r_2(p_{gd} - x_{id}) \quad (1)$$

The new velocity value is then used to calculate the next position of the particle in the search space, according to the following position update equation:

$$x_{id} = x_{id} + v_{id} \quad (2)$$

This process is then iterated until some stopping criterion is satisfied.

Here $X_i = (x_{i1}, \ldots, x_{iD})$ represents the position of the ith particle in a D-dimensional search space, $P_{besti} = (p_{i1}, \ldots, p_{iD})$ is i^{th} particle's pbest position, $P_{gbest} = (p_{g1}, \ldots, x_{gD})$ is gbest position and $V_i = (v_{i1}, \ldots, v_{iD})$ is the velocity of i^{th} particle. The inertia component serves as the memory of previous velocity, cognition component tells about the personal experience of the particle and the social component represents the cooperation among particles. Acceleration constants c_1, c_2 and inertia weight w are predefined by the user and r_1 and r_2 are the uniformly generated random numbers in the range [0, 1].

In basic PSO search for global optimum solution is accomplished by maintaining a dynamic balance between exploration and exploitation. Exploration is the ability of an algorithm to explore different regions of the search space in order to locate a good optimum. Exploitation, on the other hand, is the ability to concentrate the search around a promising region in order to refine a candidate solution [6]. PSO optimally balances these contradictory objectives by the use of velocity update equation. The social component of velocity update equation is responsible for exploration while the cognitive component is responsible for exploitation ability of PSO.

3 Multi-swarm PSO (MSPSO)

The main idea behind the proposed parallelization strategy is to first explore the search space in order to identify regions with high quality solutions and then to refine these high quality solutions. The proposed algorithm decomposes the search process to work in two stages. In the first stage of the algorithm the search space is explored by employing multiple independent swarms in parallel. So we call this stage the parallel stage of the algorithm. In the second stage of the algorithm the good solutions obtained at the parallel stage are refined by a single swarm. So we call it the sequential stage of the algorithm. Figure 1 shows the basic idea that stands behind our algorithm. The algorithm starts with multiple independent swarms with almost equal number of particles. There are as many processes as there are swarms, one process to perform calculations for each swarm. The data and calculations are independent on all processes so different parameters can be used for different swarms even if they are not experimented (except for the number of particles) in this paper. Each process runs a PSO with its own swarm without any communication with other swarms until termination and then sends its best and second best particles with function values to the 0 process. Here the parallel stage of the algorithm ends and the sequential stage begins when the process 0 receives the best and the second best particles of all the swarms including itself. The received particles constitute the initial swarm for a new PSO which is now run on the process 0. Then the best solution obtained by this process is the result of the algorithm.

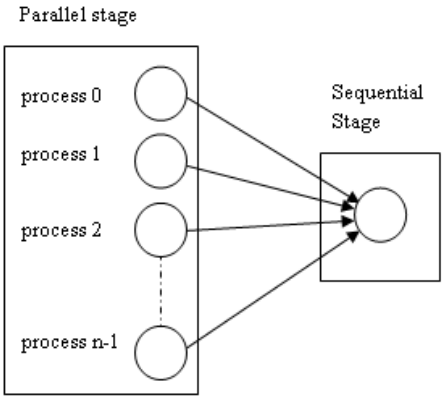


Figure 1: Diagrammatic representation of MSPSO.

Algorithm:

1. ***Parallel Stage***

(a) *Set the number of processes np and assign each process its rank r.*

(b) *Initialize for every process*
The set of parameters c_1, c_2, w, number of particles (n_r), maximum number of iterations (n_it), dimension of search space (D) and the range for each decision variable.
Randomly initialize the positions and velocities for each particle ($i = 1, 2, \ldots, n_r$).

(c) *Optimize for every process*
Evaluate function value for each particle.
Find the personal best position for each particle and the best position of the own swarm.

(d) *Update for each process velocities and positions of each particle using the equations (1) and (2) respectively.*

(e) *If the stopping criteria is satisfied go to step (f) else repeat steps (c) and (d) above until stopping criterion is met and then go to step (f).*

(f) *For each process find the second best particle of the swarm.*

2. ***Communication step***

For each process send the best and second best particles along with their function values to the process 0.

3. ***Sequential Stage***

For the process 0

(a) *Receive the best and second best particles along with their function values from all processes including itself.*

(b) *Initialize:*
Set the received particles' positions as the initial particles' positions for a new PSO.
Randomly initialize the velocities for each particle ($j = 1, 2, \ldots, 2 * np$).

(c) *Optimize the particle swarm and update velocities and positions of particles according to equations (1) and (2) respectively.*

(d) *If the termination condition is satisfied go to step (e) below, else repeat the procedure until termination criterion is met and then go to step (e) below.*

(e) *Report the best particle found in the sequential stage. This is the solution obtained by the algorithm.*

In the parallel stage of this algorithm each process executes its own copy of sequential PSO without communicating with other processes, i.e., the swarms have no knowledge of the other swarms' best particles. This means that the global nature of the sequential PSO is not maintained in the parallel stage of MSPSO.

4 Experiments and discussions

This section focuses on the efficacy of the proposed parallel version against the sequential PSO as tested on benchmark functions. To avoid attributing the results to the choice of a particular initial population and to conduct fair comparisons, we have taken 100 trials for each test, starting from various randomly selected points in the search space.

4.1 Experimental setup

Both the sequential and the parallel implementations of PSO have been experimented on a multiuser LINUX cluster having 3 identical HP ProLiant DL140G2 nodes dedicated to computations. The configuration of each node is as follows:
Processors 2 x Xeon 3.4 GHz (800 MHz FSB), 1024 KB L2 Cache, 4 x 512 PC2-3200 DDR2 memory, 2 x 146 GB ULTRA320 10k NHP, Intel Chipset is E7520, Integrated Dual Broadcom 10/100/1000, Integrated 10/100 NIC dedicated for server management, KVM PS/2 Console/0x1x8 KVM Console Switch with CAT5e cables, Interconnect switch- Gigabit Manageable Layer 2 Switch (DX-5024GS), Operating System-Red Hat Enterprise Linux Rel. 3.0.
Application code is written in C using MPI library for parallelization.

4.2 Selection of parameters

In case of many algorithms values of some parameters have to be provided by the user. PSO also has some parameters. In literature, different values of these parameters have been used. Here we use $w = 0.5, c_1 = c_2 = 2$. In order to make fair comparisons the size of population is always equivalent between the compared algorithms for a given problem. For the sequential PSO the swarm size was taken to be 100 and stopping criterion was to converge to a solution within tolerance limit (0.001 for test problems) or exceeding maximum number of iterations (1000 for test problems). For the multi-swarm PSO, in the parallel stage the swarm sizes for initial swarms were taken according to the following formula:

$$n_r = floor((r + 1) * 100/np) - floor(r * 100/np)$$

Where the symbols have the same meaning as given in the algorithm. The stopping criteria used for parallel and sequential stage are same as that for sequential PSO with the only difference that the number of iterations are different for parallel and sequential stage and for different problems. Experiments were performed for many different combinations of number of iterations at parallel and sequential stage. Finally we used the ones (Table 1) that yielded the best results for considered test functions. All the other PSO parameters used in the parallel and sequential stages of the multi-swarm PSO were same as those for sequential PSO.

Table 1: Maximum number of iterations used for test functions

Function	Maximum number of iterations	
	Parallel Stage	Sequential Stage
Sphere	300	300
Ackley	600	300
Griewank	600	300
Schwefel 3	700	300

4.3 Performance measures for parallel algorithms

Performance measures for parallel algorithmsIn order to measure the performance of the proposed parallel algorithm some performance measures have been borrowed from literature [1]. The definitions of these performance measures as given in literature are as follows:

4.3.1 Speedup

If T_k be the execution time of an algorithm using k processors. Then speedup computes the ratio between the mean execution time $E[T_1]$ on one processor and mean execution time $E[T_k]$ on k processors because of its nondeterministic nature.

Speedup due to k processors

$$s_k = \frac{E[T_1]}{E[T_k]}$$

4.3.2 Efficiency

Efficiency is the normalized version of speedup. It normalizes the speedup value to a certain percentage and is given by

$$e_k = \frac{s_k}{k} \times 100\%$$

4.4 Testing with benchmark functions

In order to test and compare the performance of different evolutionary optimization methods a suite of mathematical benchmark problems is often used. Out of these, 4 important problems have been selected to test the efficiency of the proposed algorithm. These problems are of continuous variables and have different degree complexity and multimodality. The selected problems include unimodal and multi modal functions which are scalable (the problem size can be varied as per the user's choice). Here problem size for all problems is kept fixed to 30. All these problems are of minimization type having their minimum at 0. These test functions are listed in Table 2.

The Table 3 provides the summary of experimentations with 2 to 6 independent swarms each swarm residing on a separate processor. The results shown for 1 processor are those obtained by the sequential PSO because we want to compare MSPSO with sequential PSO. Also since the definition of speedup requires that the sequential and parallel algorithms must be compared by running them until the solution of same quality has been found. So in order to make fair comparisons the execution times shown in the table are mean execution times of those runs (out of 100) which ended in a solution within the tolerance limit. For all the problems, the solutions within the tolerance limit (0.001) were obtained in atleast one of the trials.

In the Figure 2 we see that as the number of processors increases the execution time begins to decrease. This is because for fixed problem size, as the number of processors employed in solving it increases, the computation time (work load) per processor decreases. And at some point, a processor's work load becomes comparable with its parallel overhead. From this point onwards, the execution time of problem starts increasing. It means that there is an optimal number of processors to be used for any given problem. This is called the parallel balance point of the problem.

In MSPSO we see that there are two causes for the increase in execution time after parallel balance point. One is the increased communication overhead with the increase in number of processors. The other is the increased idle time of the processes (other than 0 process) in sequential stage due to increasing swarm size at this stage with the increasing number of processes. Parallel balance point, as obtained by our experiments, for the Sphere, Ackley, Griewank and Schwefel 3 functions are 4, 2, 2 and 4 respectively. At the parallel balance point a significant speedup is achieved for all the four test problems. Clearly the results for test problems obtained by MSPSO show its success in terms of speed and accu-

Table 2: Formulae for test functions used

Sl.	Name	Function	Bounds
1	Sphere	$\sum_{i=1}^{n} {x_i}^2$	$[-5.12, 5.12]^n$
2	Ackley	$-20\exp(-0.02\sqrt{\frac{1}{n}\sum_{i=1}^{n} {x_i}^2}) - \exp(\frac{1}{n}\sum_{i=1} n\cos(2\pi x_i)) + 20 + e$	$[-30, 30]^n$
3	Griewank	$1 + \frac{1}{4000}\sum_{i=1}^{n} {x_i}^2 - \prod_{i=1}^{n}\cos(\frac{x_i}{\sqrt{i}})$	$[-600, 600]^n$
4	Schwefel 3	$\sum_{i=1}^{n} \lvert x_i\rvert + \prod_{i=1}^{n} \lvert x_i\rvert$	$[-10, 10]^n$

Table 3: Experimental results for test functions[a]

Number of processors → / Functions↓		1	2	3	4	5	6
Sphere	E T (sec.)	0.158367	0.097974	0.083315	0.070784	0.088169	0.094499
	Speedup		1.616429	1.90082	2.237349	1.79619	2.038887
	Efficiency		0.808215	0.633607	0.559337	0.359238	0.339815
Ackley	E T (sec.)	0.476015	0.277418	0.276438	0.298943	0.325905	0.331064
	Speedup		1.715881	1.721962	1.592331	1.460598	1.437835
	Efficiency		0.85794	0.573987	0.398083	0.2921	0.239639
Griewank	E T (sec.)	0.431550	0.248656	0.255707	0.270887	0.290346	0.306268
	Speedup		1.73553	1.687674	1.593105	1.486331	1.40906
	Efficiency		0.867765	0.562558	0.398276	0.297266	0.234843
Schwefel 3	E T (sec.)	0.332538	0.294888	0.252413	0.220606	0.265889	0.267801
	Speedup		1.127675	1.317436	1.507387	1.250668	1.241738
	Efficiency		0.563838	0.439145	0.376847	0.250134	0.206956

[a]Here E T stands for execution time.

racy.

5 Application to Lennard-Jones Problem

5.1 Problem Description

The molecular conformation problem consists of finding a configuration of atoms in a cluster or molecule whose potential energy is minimum. It is a central problem in the study of cluster statics, or the topography of a potential energy function in an internal configuration space. This problem is also important in the study of molecular dynamics, in which its solution is thought to provide the zero-temperature configuration or ground-state of the molecule. From the viewpoint of mathematical optimization, it is a difficult global optimization problem which does not yield easily either to discrete or to continuous optimization methods [2, 5]. In its simplest form it is called Lennard-Jones problem. Even in this case, the problem of finding a global minimum of the energy can be extremely difficult due to the excessive number of non-global minima.

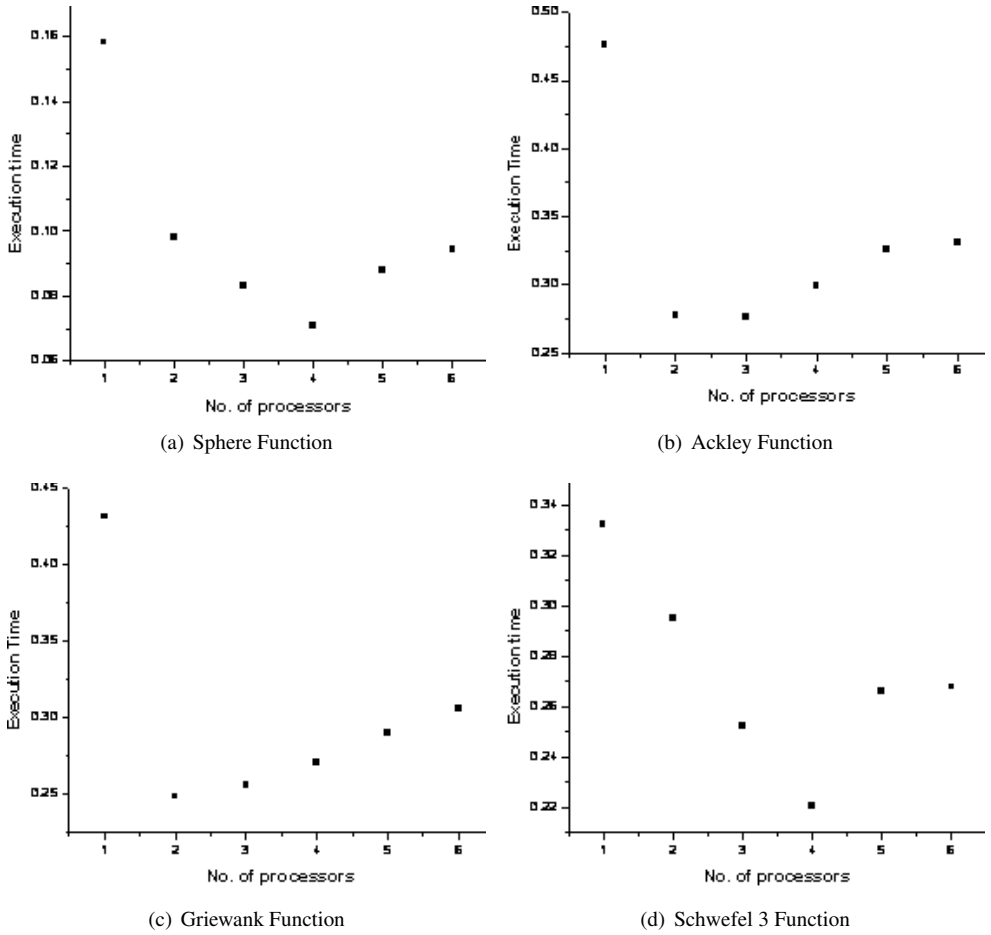
The Lennard-Jones problem assumes that the potential energy of a cluster of atoms is given by the sum of pairwise interactions between atoms, with these interactions being Vander Waals forces given by the Lennard-Jones 6-12 potential. The problem consists of determining the positions of an n atom cluster in such a way that the Lennard-Jones potential (LJP)

$$V = \sum_{i=1}^{n-1}\sum_{j=i+1}^{n} (r_{ij}^{-12} - 2r_{ij}^{-6}) \tag{3}$$

generated by atomic interactions is minimized, where r_{ij} is the Euclidean distance between the points t_i and t_j . Now since each point corresponds to Cartesian coordinates in each of the x, y and z directions so the actual number of variables is three times the number of atoms in the given cluster. The problem is then to find the positions of each atom of the cluster that corresponds to the global minimum value of V, equation (3).

5.2 Application of MSPSO to L-J problem

Our purpose of applying the MSPSO to Lennard-Jones problem is twofold. The first is that we want to reproduce the minimum energies for 8 and 9 atom clusters using the newly developed algorithm. The other is that we want to test its capabilities by studying its behavior when applied to this challenging problem. Here its capability to seek out the global minimum in a function with large number of local minima will be severely tested. If it can accomplish this test as well, then it should merit recognition as a valuable tool for treating other global optimization problems. As far as we are



(a) Sphere Function (b) Ackley Function

(c) Griewank Function (d) Schwefel 3 Function

Figure 2: Execution time versus number of processors for test problems.

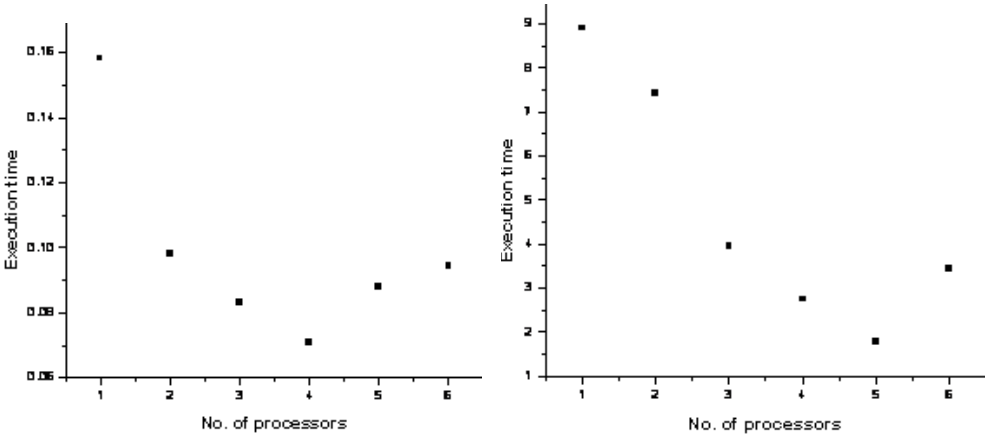
aware this approach has never been used till date, for the solution of L-J problem.

Table 4: Maximum number of iterations used for LJP

Function	Maximum number of iterations	
	Parallel Stage	Sequential Stage
LJP 8 atoms	3000	100
LJP 8 atoms	3000	100

The problem is to find the most stable conformation of the clusters with 8 and 9 atoms which have known global minimum energy values -19.821489 and -24.113360 respectively [7]. Cartesian coordinates of each atom are considered, the search space for all the atoms is given by $[-2, 2]^3$ in case of 8 and 9 atoms respectively. The tolerance limit for both the cases is set to be .000001. All other parameters except the maximum number of iterations (10000 for sequential PSO) were same as those used for benchmark test problems as described in subsection 5.2 of this paper. In both the cases (8 and 9 atoms) the PSOs at the parallel and the sequential stage were run until the tolerance limit or the maximum number of iterations (Table 4) is reached. We repeated the runs 100 times and recorded the execution times of those runs which ended in a solution within the tolerance limit. For both problems the known minimum values were obtained in atleast one of the 100 runs.

The numerical and graphical results for both the cases are shown in the Table 5 and Figure 3. The parallel balance point for 8 and 9 atoms problems are 4 and 5 respectively. At the parallel balance point the efficiency is nearly equal to 1 (the ideal value of efficiency) which shows very high performance of MSPSO at this point. This is due to the large number of iterations in parallel stage and a small number of iterations in the sequential stage. Since most of the search is completed in the parallel stage and afterwards the very good solutions ob-



(a) LJP 8 atoms (b) LJP 9 atoms

Figure 3: Execution time versus number of processors for L-J Problem.

Table 5: Experimental results for test functions.
a

Number of processors → / Functions↓		1	2	3	4	5	6
LJP 8 atoms	E T (sec.)	6.047346	5.951891	3.183264	1.540019	2.750495	3.40844
	Speedup		1.016038	1.899731	3.9268	2.198639	1.774227
	Efficiency		0.508019	0.633244	0.9817	0.439728	0.295704
LJP 9 atoms	E T (sec.)	8.909537	7.416846	3.954065	2.746731	1.786179	3.423559
	Speedup		1.201257	2.25326	3.243687	4.988043	2.60242
	Efficiency		0.600628	0.751087	0.810922	0.997609	0.433737

[a]Here E T stands for execution time.

tained at the parallel stage are refined at the sequential stage in a few iterations, so the idle time of the processes (other than 0 process) at the sequential stage is very small. This is the main reason for high efficiency of MSPSO for L-J problem.

From the simulation results it is evident that the proposed approach is a definite improvement in terms of speedup. The speedup of the algorithm with a large number of processes, for larger number of atoms in Lennard-Jones problem and for some more test problems will be investigated soon.

6 Conclusions

This paper presented a parallel global optimization algorithm based on PSO and tested it with several test problems and successfully applied for solving Lennard-Jones problem for clusters containing 8 and 9 atoms. Although the works containing application of PSO for solving this problem exist in literature [3], this is for the first time that parallel PSO has been applied to the Lennard-Jones potential problem. The results show that the solutions produced by parallel PSO are as good as those produced by the sequential PSO and that the algorithm achieves a substantial speedup. Therefore MSPSO can be used as a general purpose global optimizer. Future direction for work would be to test the performance of the proposed algorithm for a larger number of processes and on other real world applications.

7 Acknowledgements

This work is supported financially by Council of Scientific and Industrial Research, India and also by Ministry of Human Resource Development, India. We are also thankful for the support provided for our work by the Institute Computer Center, Indian institute of Technology Roorkee, India.

References

[1] Alba, E. Parallel evolutionary algorithms can achieve super-linear performance. *Information Processing Letter*, 82(1):7-13, 2002.

[2] Bernard, R.B., Bruccoleri, R.E., Olafson, B.D., States, D.J., Swaminathan, S., and Karplus, M.

CHARMM: A program for macromolecular energy, minimization, and dynamics calculations. *Journal of Computational Chemistry*, 4:187-271, 1983.

[3] Call,S.T., Zubarev, D.Y., and Boldyrev, A.I. Global Minimum Structure Searches via Particle Swarm Optimization. *Journal of Computational Chemistry*, 28:1177-1186, 2007.

[4] Chu, S.C. and Pan, J.S. Intelligent parallel particle swarm optimization algorithms. *Studies in Computational Intelligence (SCI)*, 22:159-175, 2006.

[5] Colvin, M., Judson, R., and Meza, J. A genetic algorithm approach to molecular structure determination. Paper presented at *International Conference on Industrial and Applied Mathematics*, Washington DC, 1991.

[6] Engelbercht, A.P. *Fundamentals of computational swarm intelligence*. John Wiley & Sons, 2005.

[7] Hoare, M.R. and Pal, P. *Adv. Phys.* 20(161),1971; *Nature (Physical Sciences)* 230(5), 1971; *Nature (Physical Sciences)* 236(35), 1972.

[8] Kennedy, J. and Eberhart, R.C. Particle swarm optimization. In *proceedings IEEE Conf. on Neural Networks*, Perth, pp.1942-1948, 1995.

[9] Kennedy, J., Eberhart, R.C., and Shi, Y. *Swarm Intelligence*. Morgan Kaufmann Publishers, San Francisco, 2001.

[10] Kim, J.Y., Jeong, H.M., Lee, H.S., and Park, J.H. PC cluster based parallel PSO algorithm for optimal power flow. In *proceedings 14th International Conference on Intelligent System Applications to Power Systems (ISAP' 2007)*, Kaohsiung, Taiwan, 2007.

[11] Koh, B., George, A.D., Haftka, R.T., and Fregly, B.J. Parallel Asynchronous Particle swarm optimization, *International Journal of Numerical Methods in Engineering*. 67:578-595, 2006.

[12] Sahin, F., Yavuz, M.C., Arnavut, Z., and Uluyo, O. Fault diagnosis for airplane engines using Bayesian networks and distributed particle swarm optimization. *Parallel Computing*, 33:124-143, 2007.

[13] Schutte, J.F., Reinbolt, J.A., Fregly, B.J., Haftka, R.T., and George, A.D. Parallel global optimization with the particle swarm algorithm. *International Journal of Numerical Methods in Engineering*, 61(13):2296-2315, 2004.

[14] Snir, M., Otto, S., Lederman, S.H., Walker, D., and Dongarra, J. *MPI: The Complete Reference*. Massachusetts Institute of Technology, Cambridge, 1996.

[15] Venter, G., Sobieski, J.S. A parallel Particle Swarm Optimization algorithm accelerated by asynchronous evaluations, *Journal of Aerospace Computing, Information, and Communication*, 3(3):123-137, 2005.

[16] Waintraub, M., Schirru, R., and Pereira, C.M.N.A. Multiprocessor modeling of parallel Particle Swarm Optimization applied to nuclear engineering problems. *Progress in Nuclear Energy*, 51:680-688, 2009.