Parallel Particle Swarm Optimization Algorithm with Applications

Emre Yilmaz

January 14, 2021

Abstract

In this paper, we develop parallel implementation of Particle Swarm Optimization (PSO) to solve computationally expensive large scale global optimization problems. We apply our algorithm to some test problems to show the efficiency.

1 Introduction

Numerical optimization has been studied for many years due to solve many problems in science, engineering and real life. These problems may have multiple minimizers. In order to solve these problems, global optimization algorithms have been developed such as particle swarm optimization (PSO) Kennedy and Eberhart (1995), genetic algorithms (GA) Deb (1999), artificial bee colony algorithm (ABC) Karaboga and Basturk (2007).

Among the other method PSO some advantages owing to its simplicity, robustness, efficiency, and global search capability. PSO was proposed by Kennedy and Eberhart in 1995 and it depends on the social behavior of bird flock or fish school Kennedy and Eberhart (1995); Eberhart and Yuhui Shi (2001). In the PSO system, a number of particles coexist and cooperate to get the optimal solution. Starting from an initial population, the algorithm consists of updating the velocity and the position of each particle in the swarm Moraes et al. (2015); He and jing Huang (2014). Similar to other heuristic algorithms, the PSO has several disadvantages such as PSO is susceptive to premature for being trapped in local minima, another one is that PSO converges slowly during the latter search process and last one is that PSO needs a long time to solve problems with time-consuming fitness functions Lai and Zhou (2019). The parallelization of PSO is an effective way to improve the performance of PSO, which has been widely used in many fields Schutte et al. (2004). The comprehensive overview can be found in the study Lalwani et al. (2019).

In this study, we present a parallel PSO algorithm to solve computationally expensive large scale global optimization problems. We apply to large scale test problems of global optimization to show the effectiveness of our algorithm.

2 Serial Particle Swarm Algorithm

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. For each particle, Kennedy and Eberhart proposed that its position x^i be updated in the following manner position xi be updated in the following manner:

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

with a pseudo-velocity v_{k+1}^i calculated as follows:

$$v_{k+1}^i = w_k v_k^i + c_1 r_1 (p_k^i - x_k^i) + c_2 r_2 (p_k^g - x_k^i) \tag{2}$$

where k indicates a (unit) pseudo-time increment, p_k^i is the best ever position of particle i at time k (the cognitive contribution to the pseudo-velocity vector v_{k+1}^i), and 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, the cognitive and social scaling parameters $c1$ and c_2 are selected as $c_1 = 1.7$ and $c_2 = 2.2$.

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 p_k^i is denoted by f_{best}^i and the best ever fitness value of the overall swarm at co-ordinates p_k^g by f_{best}^g. At time step $k = 0$, the particle velocities v_0^i are initialized to values within the limits $0 \leq v_0 \leq v_0^{max}$. The vector v_0^{max} is calculated as a fraction of the distance between the upper and lower bounds $v_0^{max} = \eta(x_{UB} - x_{LB})$, with $\eta = 0.5$. With this background, the PSO algorithm is described as follows:

1. Initialize

(a) Set constants k_max, c_1, c_2, K, 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 $x_0^i \in D$ in R^n for $i = 1, \ldots, p$

(e) Randomly initialize particle velocities $0 \leq v_0^i \leq v_0^{max}$ for $i = 1, \ldots, p$

(f) Evaluate fitness values f_0^i using design space co-ordinates x_0^i for $i = 1, \ldots, p$

(g) Set $f_{best}^i = f_0^i$, $p^i = x_0^i$ for $i = 1, \ldots, p$

(h) Set f_{best}^g to best f_{best}^i and g_0 to corresponding x_0^i.

2. Optimize

(a) Update particle velocity vector v_{k+1}^i using Eqn. 2

(b) Update particle position vector x^i_{k+1} using Eqn. 1

(c) Update dynamic maximum velocity v^{max}_k and inertia w_k

(d) Evaluate fitness value f^i_k using design space co-ordinates x^i_k

(e) If $f^i_k \leq f^i_{best}$ then $f^i_{best} = f^i_k$, $p^i = x^i_k$

(f) If $f^i_k \leq f^g_{best}$ then $f^g_{best} = f^i_k$, $p^g = x^i_k$

(g) If f^g_best 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^{max}_{k+1} 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

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.

3 Parallel Particle Swarm Algorithm

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 v^i_{k+1} using Eqn. 2

(b) Update particle position vector x^i_{k+1} using Eqn. 1

(c) Concurrently evaluate fitness values f^i_k using design space co-ordinates x^i_{k+1} for $i = 1, \ldots, p$

(d) If $f^i_{k+1} \leq f^i_{best}$ then $f^i_{best} = f^i_{k+1}$, $p^{i+1} = x^i_{k+1}$ for $i = 1, \ldots, p$

(e) If $f^i_{k+1} \leq f^g_{best}$ then $f^g_{best} = f^i_{k+1}$, $p^g = x^i_{k+1}$ for $i = 1, \ldots, p$

(f) If $f_b^g est$ 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 k

(l) Go to 2 (a)

The above logic is illustrated as a flow diagram in Figure 1 without detailing the working of the dynamic reduction parameters.

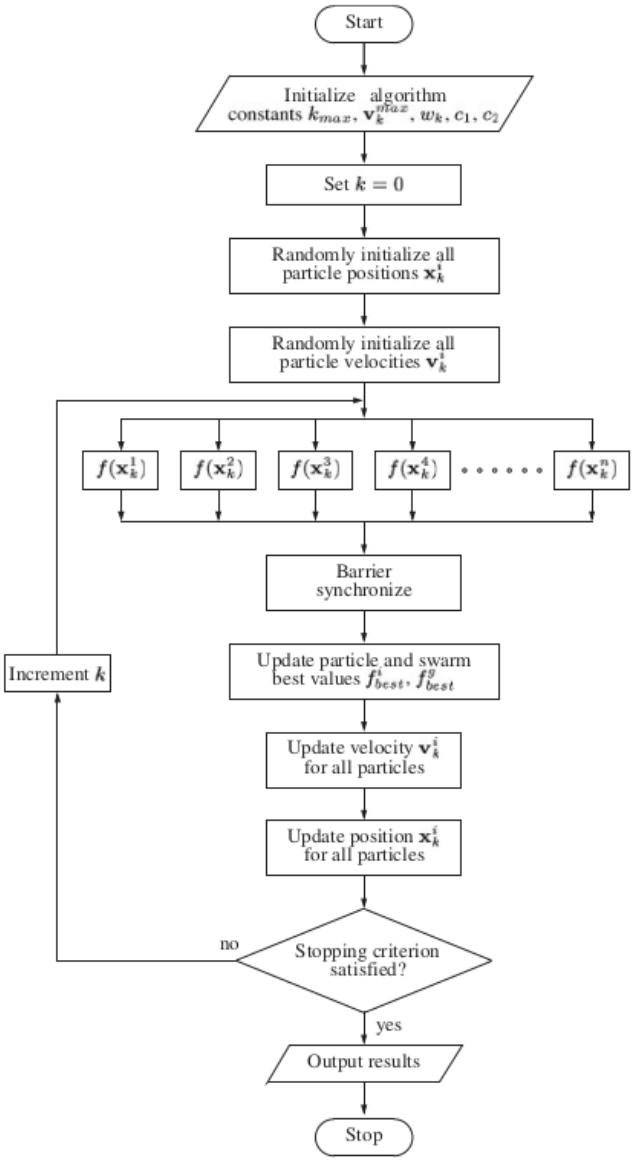

Figure 1: The flowchart of parallel PSO algorithm.

4 Numerical Examples

In this section, we present the results of "Serial" and "Parallel" applications of PSO to 3 different test problems of Booth, Beale and Ackley. We compare the results with each other.

We apply the "Serial" and "Parallel" applications of PSO by using C++ and OPENMP framework on PC with configuration of Intel Core i5-8300H, 16GB RAM. The numerical results are reported in Table 1. In the table below, we report the computational time, speed up and efficiency.

Table 1: The numerical result

	Booth (MS)	Beale (MS)	Ackley (MS)
Serial 200 Particle	8492	17416	13520
Parallel 200 Particle 5 Threads	3266	6582	5041
Speedup (TS/TP)	2,60	2,64	2,68
Efficiency (Speedup/Thread Count)	0,52	0,528	0,536

5 Conclusion

In this study, we present both serial and parallel implementation of PSO. We apply them to some test problems in the literature. We present the results obtained from these implementations. It is observed that the Parallel PSO is faster than Serial PSO.

References

Deb, K. (1999). An introduction to genetic algorithms. *Sadhana*, 24(4):293–315.

Eberhart and Yuhui Shi (2001). Particle swarm optimization: developments, applications and resources. In *Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546)*, volume 1, pages 81–86 vol. 1.

He, G. and jing Huang, N. (2014). A new particle swarm optimization algorithm with an application. *Applied Mathematics and Computation*, 232:521 – 528.

Karaboga, D. and Basturk, B. (2007). A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm. *Journal of Global Optimization*, 39(3):459–471.

Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Proceedings of ICNN'95 - International Conference on Neural Networks*, volume 4, pages 1942–1948 vol.4.

Lai, X. and Zhou, Y. (2019). An adaptive parallel particle swarm optimization for numerical optimization problems. *Neural Computing and Applications*, 31(10):6449–6467.

Lalwani, S., Sharma, H., Satapathy, S. C., Deep, K., and Bansal, J. C. (2019). A survey on parallel particle swarm optimization algorithms. *Arabian Journal for Science and Engineering*, 44(4):2899–2923.

Moraes, A. O., Mitre, J. F., Lage, P. L., and Secchi, A. R. (2015). A robust parallel algorithm of the particle swarm optimization method for large dimensional engineering problems. *Applied Mathematical Modelling*, 39(14):4223 – 4241.

Schutte, J. F., Reinbolt, J. A., Fregly, B. J., Haftka, R. T., and George, A. D. (2004). Parallel global optimization with the particle swarm algorithm. *International journal for numerical methods in engineering*, 61(13):2296–2315. 17891226[pmid].