Parallelization of Particle Swarm Optimization using Message Passing Interfaces (MPIs)

Gagan Singhal, Abhishek Jain
Department of Electronics and Computer Engineering,
IIT Roorkee, Uttarakhand – 247 667, India
singhalgagan1@gmail.com
abhi111jain@gmail.com

Amalendu Patnaik
Department of Electronics and Computer Engineering,
IIT Roorkee, Uttarakhand – 247 667, India
apatnaik@ieee.org

Abstract: **Motivated by the growing demand of accuracy and low computational time in optimizing functions in various fields of engineering, an approach has been presented using the technique of parallel computing. The parallelization has been carried out on one of the simplest and flexible optimization algorithms, namely the particle swarm optimization (PSO) algorithm. PSO is a stochastic population global optimizer and the initial population may be provided with random values and later convergence may be achieved. The use of message passing interfaces (MPIs) for the parallelization of the asynchronous version of PSO is proposed. In this approach, initial population has been divided between the processors chosen at run time. Numerical values obtained using above approach are at last compared for standard test functions.**

Keywords: asynchronous PSO, parallel computing, message passing interfaces

I. Introduction

Today one of the major challenges in the engineering and scientific world is finding accurate results of a problem and those too with minimum computational time. The solution to complex engineering problems is not straightforward and usually requires an iterative process which would yield the result in a number of runs i.e. the solution is approached a step further in every iteration. The accuracy of the result is determined by the maximum permissible iterations or the minimum amount of error, which would serve the purpose. Several algorithms have been proposed in order to optimize the solutions to these engineering problems. Evolutionary computational methods such as the genetic algorithm (GA), ant colony optimization (ACO) and particle swarm optimization (PSO) are efficient solutions to the necessary requirements [1].

One of the major drawbacks of these optimization techniques using evolutionary computational methods is their high computation time, because of their iterative nature. Parallelization of some of these methods can reduce the time of computation drastically. In this paper, we have implemented the parallelization of the asynchronous version of PSO using Message Passing Interfaces (MPIs) [2]. The developed technique is tested for several standard optimization functions to achieve their minima within tolerance limits.

The next section of the paper describes the PSO algorithm in brief. In Section-III we have described the parallel PSO algorithm using MPIs. Results of implementation of this algorithm for several test functions are presented in Section-IV, followed by the conclusion.

II. Particle Swarm Optimization

Particle Swarm Optimization (PSO) [3, 4] is an algorithm which derives its inspiration from the social behavior and dynamics of insects, birds and fish and has a performance comparable to that of GAs. These breeds of animals are very intelligent in the way they optimize the conditions for protection from predators, seek food and mates etc. If they are left in an initialized situation randomly they automatically adjust so as to optimize their surroundings. This leads us to the stochastic character of the PSO.

In analogy to the birds here a number of agents are considered each been given a particle number i and each agent possessing a position defined by coordinates in n-dimensional space. These particles/agents also possess an imaginary velocity which in turn reflects the proximity to the optimal position. The initialization is random and thereafter a number of iterations are carried out with the particle velocity (v) and position (x) updated at the end of every iteration. They are updated as follows:

Position: $$x_{iter+1}^{\ i} = x_{iter}^{\ i} + v_{iter+1}^{\ i} \quad (1)$$

Velocity: $$v_{iter+1}^{\ i} = w^i v_{iter}^{\ i} + c_1 r_1(x_{best}^{\ i} - x_{iter}^{\ i}) + c_2 r_2(x_{gbest} - x_{iter}^{\ i}) \quad (2)$$

where

w^i:	inertia possessed by each agent
$x_{best}^{\ i}$:	most promising location of the agent
x_{gbest}:	most promising location amongst the agents of the whole swarm
c1:	cognitive weight which represents the private thinking of the particle itself. It is assigned to particle best $x_{best}^{\ i}$
c2:	social weight assigned to swarm best x_{gbest} which represents the collaboration among particles.

978-1-4244-5612-3/09/$26.00 ©2009 IEEE

Authorized licensed use limited to: TRINITY COLLEGE LIBRARY DUBLIN. Downloaded on July 07,2020 at 16:14:41 UTC from IEEE Xplore. Restrictions apply.

III. PARALLEL PSO

Parallel computing is a technique which can change the face of the efficiency of PSO algorithm in a tremendous manner. The algorithm can be implemented using multiple processors which multiply the running speed by a large factor and the result can be obtained much earlier at a faster rate. Hence more and more complex functions may be optimized using PSO on parallel computing mode.

Running PSO in a parallel processing mode can be easily implemented by using a language independent communication protocol called MPI. Libraries containing useful functions are available for different programming languages such as FORTRAN, C, C++ etc. The MPI is basically used in computer clusters and supercomputers. High performance is obtained using this protocol besides a very useful advantage of easy portability. It supports both point to point as well as collective communication between processors. A communicator serves as an interface through which processors communicate with each other in an MPI session. MPI provides additional features of derived data types in which user defined data types may be created. In other words MPI is a tool which forms the backbone of parallel computing when it comes to taking factors like speed and efficiency into account.

Parallel implementations of asynchronous PSO have been attempted previously by other authors [5, 6]. But in the present work, in order to parallelize PSO, the most basic Master-Slave [5] approach is not adopted where one processor is allocated only for communication with slave processors. In the present work the particles are distributed over the given no. of processors ensuring that each processor has almost equal amount of load so that the result from each processor arrives at almost the same time and also acts as a savior of time. When the best results have been calculated by the processors amongst their respective particles, then the final optimum result (of a particular iteration) is obtained and this optimum is reflected to all the processors, both the things been done by a single MPI function MPI_Allreduce() [2]. Also if there is a need to reflect some value from any processor (called root processor) among all processors, MPI_Scatter () [2] function may be used. Moreover this root processor is dynamic i.e. any processor which contains the best particle values near the end of iteration (before updating velocity and position) becomes the root processor. This is how processors communicate with each other and the optimum result obtained in the end.

1. Initialize for every process

(a) Constants $iter_{max}$ (maximum iterations), dim (dimension of each particle), c1, c2, v_0^{max}, x_0^{max}, w, n (total number of particles), range of initialization for asynchronous version(**R**). (C1 and c2 are usually set to 2).

(b) Set gbest_id = 0(particle 0 initially for every process).

(c) Allocate number of processes (p), and give each process its 'id'.

(d) Calculate number of particles for each process, g = floor ((id+1)*n/p) - floor (id*n/p).

(e) Randomly initialize particle positions $x_0^i \in \mathbf{R}$ for i = 1...g.

(f) Randomly initialize particle velocities $0 < v_0^i <$ v0max for i = 1...g.

(g) Set iter = 1, i=1.

2. Optimize for every process

(a) Evaluate function value f_{iter}^i using design space coordinates for i = 1...g.

(b) If $f_{iter}^i < f_{best}^i$, then $f_{best}^i = f_{iter}^i$ and $x_{best}^i = x_{iter}^i$ for i = 1...g.

(c) If $f_{best}^i < f_{best}^{gbest\_id}$, then gbest_id = i.

3. Interaction between processors

(a) Obtain gbest_all = min ($f_{best}^{gbest\_id}$ of every process).

(b) Introduce a position variable (tx) holding the position of the globally best particle (gbest_all).

4. Update

(a) Update particle velocity v_{iter}^i using (2)

(b) Update position using x_{iter}^i using (1)

5. Scrutinize

(a) If the stopping criterion is achieved, go to 6.

(b) Else i++. If i > g, then increment iter and set i = 1.

(c) Go to 2(a).

6. Report results and solutions.

The flow of the algorithm is illustrated in Figure. 1.

In this work, all computations are performed on a multiuser HP DL140G2 cluster. The Cluster configuration is CPU-AMD opteron 844/2.6*2 GHZ processor (Dual) capable of guard processing (supplied with two processor) 2 MB L2 cache/800 MHZ FSB, Memory: 2 GB 333 MHZ DDR RAM upgradable up to 32MB, HDD SCSI ultra 320 dual channel 2*144 GB 10 k (hot plug)

IV. RESULTS AND DISCUSSION

Several example optimization problems were attempted for solution using parallel PSO algorithm. Table-1 lists those testing functions and their initialization ranges of populations are listed in Table-2. Each subsequent table (Table 3 - 8)

Authorized licensed use limited to: TRINITY COLLEGE LIBRARY DUBLIN. Downloaded on July 07,2020 at 16:14:41 UTC from IEEE Xplore. Restrictions apply.

shows results for a testing function. These tables also show the number of successes in 100 runs. A decrease in elapsed time and an increase in successful runs were obtained with the increase in number of processors.

Table 1: Testing Functions

Function	Mathematical Representation
Sphere	$\sum_{i=1}^{n} x^{2}$
Rosenbrock	$\sum_{i=1}^{n-1}\left[100(x_{i+1}-x_i^{2})^{2}+(x_i-1)^{2}\right]$
Rastrigin	$\sum_{i=1}^{n}\left[x_i^{2}-10\cos(2\pi x_i)+10\right]$
Griewank	$\frac{1}{4000}\sum_{i=1}^{n}x_i^{2}-\prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right)+1$
Schaffer's f6	$0.5+\frac{\sin^{2}\sqrt{x_1^{2}+x_2^{2}}-0.5}{\left[1+0.001\left(x_1^{2}+x_2^{2}\right)\right]^{2}}$
Ackley	$-20\exp\left(-0.2\sqrt{\left(\frac{1}{n}\sum_{i=1}^{n}x_i^{2}\right)}\right)$ $-\exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos 2\pi x_i\right)+20+e$

Table 2: Function Domains

Function	Range of Search	Initialization Range
Sphere	$[-100, 100]^n$	$[50, 100]^n$
Rosenbrock	$[-100, 100]^n$	$[20, 50]^n$
Rastrigin	$[-10, 10]^n$	$[2.56, 5.12]^n$
Griewank	$[-600, 600]^n$	$[30, 100]^n$
Schaffer's f6	$[-100, 100]^n$	$[15, 30]^n$
Ackley	$[-100, 100]^n$	$[50, 100]^n$

Table 3: Results for Sphere Function (Optimal Value = 0)

Population Size	No. of Processors	No. of Successes in 100 runs	Time Elapsed (sec.)
30	1	100	0.062191
	2	100	0.036535
60	1	100	0.109622
	2	100	0.063144
120	1	100	0.214278
	2	100	0.120354
240	1	100	0.417996
	2	100	0.244677
	4	100	0.237600

Table 4: Results for Rosenbrock Function (Optimal Value = 0)

Population Size	No. of Processors	No. of Successes in 100 runs	Time Elapsed (sec.)
30	1	2	0.151956
	2	2	0.091893
60	1	3	0.298568
	2	4	0.168217
120	1	3	0.594625
	2	5	0.325775

Table 5: Results for Rastrigin Function (Optimal Value = 0)

Population Size	No. of Processors	No. of Successes in 100 runs	Time Elapsed (sec.)
30	1	8	0.191926
	2	38	0.100675
60	1	20	0.396086
	2	45	0.239695
	4	66	0.223699
120	1	62	0.763859
	2	68	0.272319
	4	77	0.232241

Table 6: Results for Griewank Function (Optimal Value = 0)

Population Size	No. of Processors	No. of Successes in 100 runs	Time Elapsed (sec.)
30	1	2	0.227252
	2	17	0.129861
60	1	2	0.454106
	2	15	0.246751
	4	31	0.236749
120	1	8	0.900212
	2	14	0.476281
	4	25	0.439019

Table 7: Results for Schaffer's f6 Function (Optimal Value = 0)

Population Size	No. of Processors	No. of Successes in 100 runs	Time Elapsed (sec.)
30	1	100	0.007281
	2	100	0.000625
60	1	100	0.008478
	2	100	0.001221

Authorized licensed use limited to: TRINITY COLLEGE LIBRARY DUBLIN. Downloaded on July 07,2020 at 16:14:41 UTC from IEEE Xplore. Restrictions apply.

Table 8: Results for Ackley Function (Optimal Value = 0)

Population Size	No. of Processors	No. of Successes in 100 runs	Time Elapsed (sec.)
30	1	6	0.232361
	2	25	0.132468
60	1	14	0.466264
	2	30	0.255247
	4	57	0.180178
120	1	14	0.925936
	2	37	0.453515
	4	66	0.230888

Note: - A run is said to be successful if the algorithm yields the function value either less than or equal to the tolerance (acceptable variation from the expected result) value taken as 0.01 for the same. Stopping Criterion in Figure 1 specifies the condition at which PSO should stop executing further i.e. either PSO has iterated for the maximum number of runs or the objective function is minimized within the tolerance limits.

V. CONCLUSION

In this paper, a generalized parallel algorithm for asynchronous PSO has been introduced. This algorithm generalizes the division of particles in the best possible manner between any number of processors allocated at the run time.

Since in parallel programming, communication between processors is a major drawback which cannot be removed completely, therefore in order to notice a reduction in time with the increase in number of processors, one has to set a fairly large problem size. This is typically the case in real world optimization problems. Along with the reduction in running time of the program, increase in the accuracy of the algorithm is observed as an added benefit of parallelization. The functions like Rosenbrock, Rastrigin, Griewank, which do not converge easily within the iterative bound of other functions like Sphere and Schaffer's f6, are observed to give more number of successful runs with multiple processors.

REFERENCES

[1]. Z.Michalewicz, D.Dasgupta, *Evolutionary Algorithms in Engineering Applications,* Springer Verlag, 1997.

[2]. Michael J. Quinn, *Parallel programming in C with MPI and OpenMP*, Tata McGraw-Hill, New Delhi, 2005.

[3]. J. Kennedy, and R. Eberhart, "Particle swarm optimization", IEEE *Int. Conf. on Neural Networks IV*, p. 1941-1948, Piscataway, NJ, 1995.

[4]. R.C. Eberhart, J. Kennedy,"A New Optimizer Using Particle Swarm Theory", *6th Int. Symp. on Micromachine and Human Science*, Nagoya, Japan, pp. 39-43, 1995.

[5]. J. F. Schutte, J. A. Reinbolt, B. J. Fregly, R. T. Haftka and A. D. George, "Parallel global optimization with the particle swarm algorithm," *Int. J. Numerical Methods Engineering*, Vol. 61, pp. 2296–2315, 2004.

[6]. G. Venter and J.Sobieszczanski-Sobieski, " A Parallel Particle Swarm Optimization Algorithm Accelerated by Asynchronous Evaluations", 6th World Congresses of Structural and Multidisciplinary Optimization, Rio de Janeiro, 30 May – 3 June, Brazil, 2005.

Authorized licensed use limited to: TRINITY COLLEGE LIBRARY DUBLIN. Downloaded on July 07,2020 at 16:14:41 UTC from IEEE Xplore. Restrictions apply.

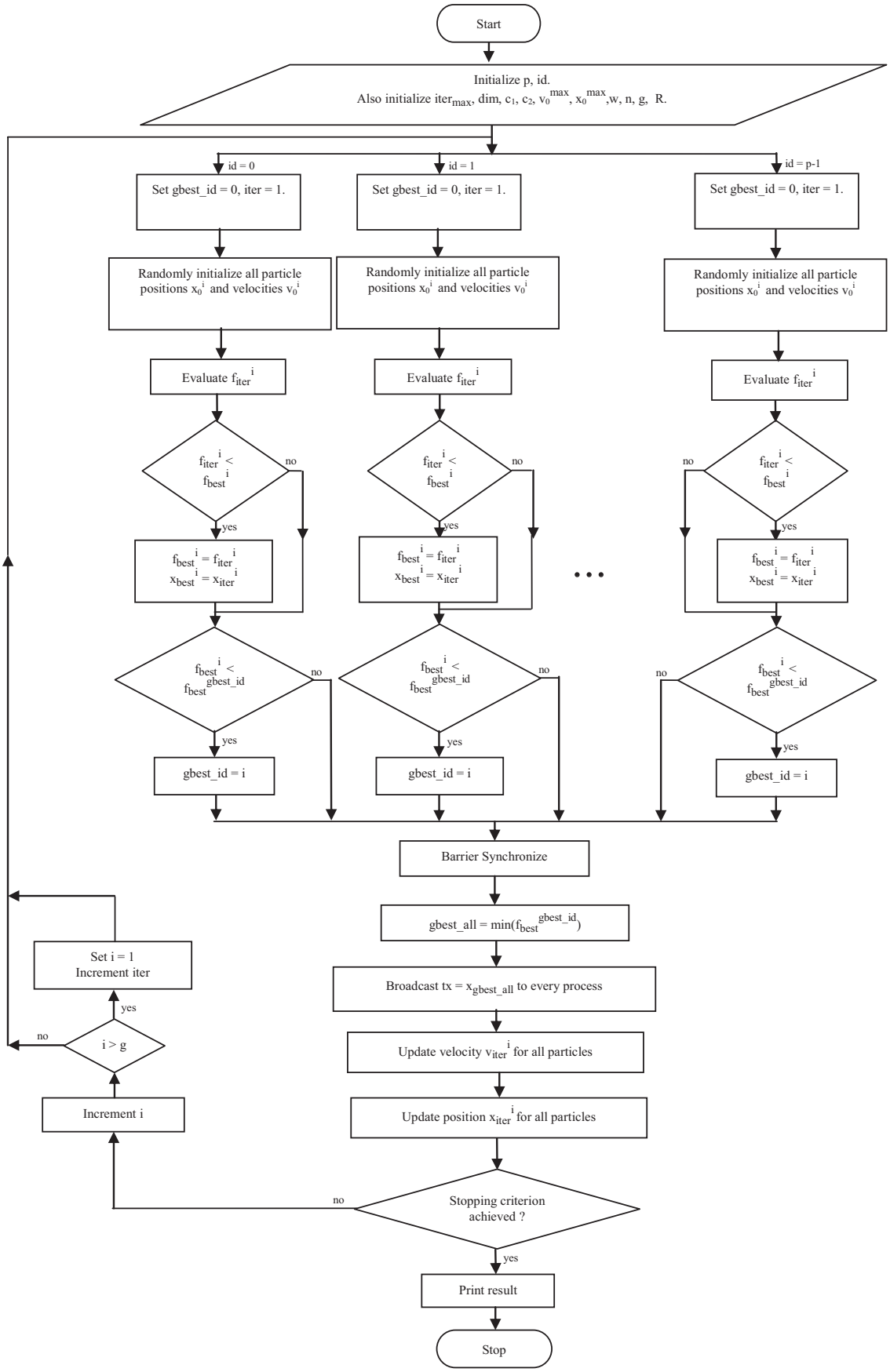

Figure. 1. Algorithm Flow

Authorized licensed use limited to: TRINITY COLLEGE LIBRARY DUBLIN. Downloaded on July 07,2020 at 16:14:41 UTC from IEEE Xplore. Restrictions apply.