IT-300 Parallel Computing
Mini Project
Parallelization of Particle Swarm Optimization

Submitted to :
Dr. Geetha V
Ms. Thanmayee

In partial fulfilment of the degree of Bachelor of Technology

Submitted by:

Gaurav Uttarkar - 16IT113
Divyam Mehta - 16IT112
Abhilash V - 16IT201
Shrinivas V Shanbhag - 16IT244

National Institute of Technology Karnataka, Surathkal
November 2018

ACKNOWLEDGEMENT

We would like to express our special thanks of gratitude to our faculty, Dr Geetha V and Ms Thanmayee as they gave us the golden opportunity, and inspired us to undertake this informative project. The completion of this project would not have been possible without their constant guidance, kind support and help. All the members have worked on bringing this project this far. We would like to extend our sincere thanks to all of them.

ABSTRACT

In this study, a sequential implementation of the Particle Swarm Optimization algorithm is converted into a concurrent version. For this reason, MPI and OpenMP libraries are operated on the parallel algorithm to make a concurrent execution on CPU.

The algorithm used here is Particle Swarm Optimization (PSO). This is a classical approach to obtain solutions for complex functions using optimization techniques.

The aim of this study is to compare serial, MPI and OpenMP implementation of the PSO algorithm as regards the average execution time of independent Monte Carlo runs and computation architecture. For this purpose, three benchmark functions are selected as test problems, and the parallel algorithm is executed on different processing units.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

Many engineering problems are based on finding the best possible solution for any specific configuration/system. The real-world engineering problems are very complicated when compared to geometric studies, the classical optimization algorithms are not able to solve these problems demanding complex mathematical computations. So ,heuristic approaches are used to get best estimate of optimum values . Genetic Algorithm , Ant Colony Optimization , Differential Evolution , and Particle Swarm Optimization (PSO) are examples of the heuristic approaches. Even though these algorithms yield better results for many engineering problems, the execution time of sequential codes or implementations restrict the research in this are as regards the time limitations. Hence, in this project , the sequential PSO algorithm is parallelized in two different manners (i.e. via different techniques aimed for deployment on different architectures), and these two parallel PSO (PPSO) algorithms are executed on both CPU and GPU using OpenMP and CUDA framework. In this project, the PPSO algorithm is implemented on different CPU and GPU architectures and the average execution time of ten independent Monte Carlo runs (executions) is compared as a performance indicator

1.1 Particle Swarm Optimization

The behavior of the algorithm greatly depends on the interactions between particles, where the members of the population are called particles (N). Each particle has velocity (vi, i=1,2,…,N) and position (xi, i=1,2,…,N). At the end of each iteration (k), the particles change their position based on their current velocity. The well-known physical position update rule, which is applied on this algorithm, is given in

$$xi[k+1] = xi[k] + vi[k]*t \quad --->(1)$$

The velocity update formulation is presented in

$$vi[k] = vi[k-1] + c1 * R * (pi[k] - xi[k]) + c2 * R * (pg[k] - xi[k]) \quad --->(2)$$

where c1 and c2 are the control parameters of the algorithm i.e . learning factors, pi is the personal best position of i'th particle, and pg is the global best position among the population, R is random number between [0-1].

The PSO algorithm can be summarized as follows:
a) the position and velocity values of each particle are assigned randomly,
b) At the beginning of iteration, cost function is evaluated (f(xi)) for each particle's position (xi),
c) Personal (the best position of each individual between the iteration 1 and k) and global best (the best position among the population at the iteration k) positions are determined based on their cost values,
d) The position of each particles is altered by using (1) and (2),
e) if the maximum number of iteration is not equal to current iteration k or minimum error criteria is not attained, then step b is revisited; else the algorithm is terminated.

1.2 Variants of PSO

The PSO algorithm with the constriction factor is a special case of the algorithm with constriction factor K.The Equation (2) changes as follows:

$$vi[k] = K*[vi[k-1] + c1 * R * (pi[k] - xi[k]) + c2 * R * (pg[k] - xi[k])\ ] \quad --->(3)$$

where $K = c1+c2$ and $K > 4$

Addition of this new parameter causes exploration and exploitation in search space. It insures convergence of particle swarm optimization algorithm.

Hierarchical PSO is known as hierarchical version of PSO called as H-PSO. In this algorithm the particles are arranged in a dynamic hierarchy. In H-PSO, a particle is influenced by its own so far best position and by the best position of the particle that is directly above it in the hierarchy.If a particle at a child node as found a solution that is better than the best so far solution of particle at parent node, then these two particles are exchanged.

Fully Informed Particle Swarm Optimization (FIPSO) use best of neighborhood velocity update strategy. In FIPSO each particle uses the information from all its neighbors to update its velocity.

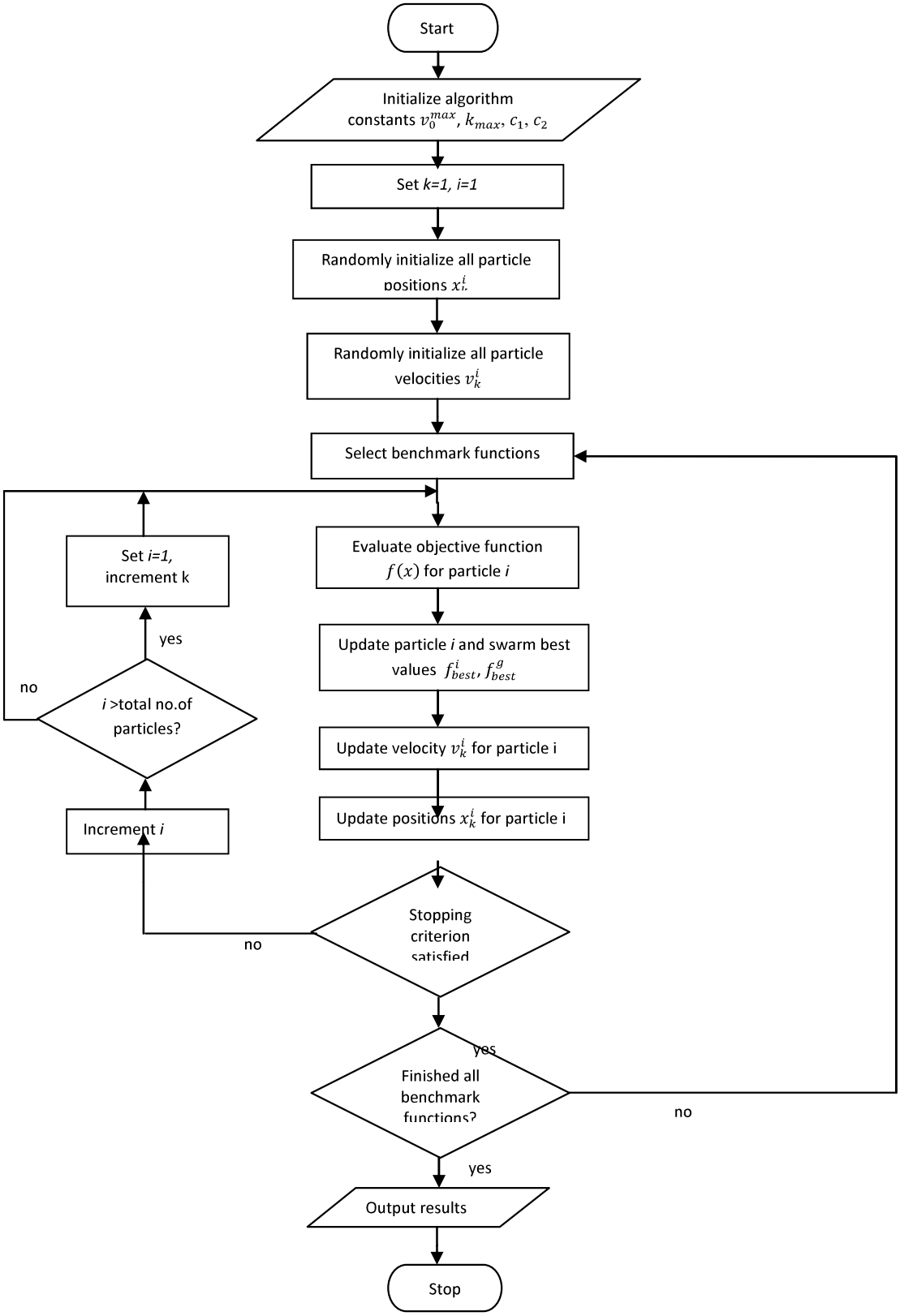


Figure 1: Flowchart of serial PSO

CHAPTER 2
LITERATURE SURVEY

Dynamic optimal reactive power dispatch based on parallel particle swarm optimization algorithm [1]

In this project, Message Passing Interface (MPI) based parallel computation and particle swarm optimization (PSO) algorithm are combined to form the parallel particle swarm optimization (PPSO) method for solving the minimisation or maximisation problems. In this method, the optimization process is implemented on a multi-processor supercomputer to reduce the computational time drastically.

MPI based parallel computation using n processors(how it works):

1. First, processor 0 reads and preprocesses all the data before scattering them to other processors' memory.
2. After processor 0's initialization work, all the processors calculate with the data that are sent by processor 0 through message passing interface.
3. When the calculations are done, results are sent back to processor 0's receiving buffer to be dealt with. Then processor 0 decides whether to continue computing or output final results and terminate program

Advantages:

1. Takes less time for computation than serial program.
2. Computation time will decrease in case of large number of processors in parallalised approach when compared with[2].

Disadvantages:

* This implementation has high message passing overheads, compared with [2].
* Only main thread does random initialisation of all particles, instead of giving each processor to do the random initialisation of their particles.

* This implementation would have used openmp along mpi for better performance in case of large number of particles.

Reference: [1]

Parallel Particle Swarm Optimization [2]

In this paper, process0 sends number of particles to be evaluated to each process. In parallel section, initialization of each particle is done, then updation of each particle position and velocity is done, then best position at all processes are taken at processor0 and best of all is considered, and will send to all the processes until stopping criteria is achieved.

Advantages:

1. Takes less time for computation than serial program.
2. All the processor does random initialisation of the particles assigned to it, thus gives better performance than [1].
3. Has less message passing over heads than [1].

Disadvantages:

1. Computation time will increase in case of large number of processors in parallelized approach when compared with[2]. Because this implementation is not using openmp along with mpi (so parallelisation is not effective).

Parallel Asynchronous PSO (PAPSO) proposed in [2] is highly scalable and more efficient than synchronous Parallel PSO when (1) heterogeneity exists in the computational task or environment, or (2) the computation-to-communication time ratio is relatively small.In PAPSO, design points in the next design iteration are analyzed before the current design iteration is completed i.e.the algorithm doesn't waits for all the analyses to complete before moving to the next design iteration,so that there are no ideal processors as one moves from one iteration to next iteration. PAPSO has lower performance when number of processors are very less and the degree of algorithm parallelism is limited by the number of particles, a problem shared with PSPSO.

Comparison of Parallel CUDA and OpenMP Implementations of Particle Swarm Optimization [3]

Since the physical constraints on micro computing devices have forced the researchers to design next generation chips , the significance of the parallelization and distributed computing grow in importance. In this paper, a sequential implementation of the Particle Swarm Optimization algorithm is converted into a concurrent version , which is executed on the cores of both CPU and GPU. For this reason , CUDA and OpenMP libraries are operated on the parallel algorithm to make a concurrent execution on CPU and GPU , respectively. The aim of this study is to compare CPU and GPU implementation of the PSO algorithm as regards the average execution time of independent Monte Carlo runs and computation architecture.

In this study, PPSO is implemented on eight different hardware devices (CPU: Intel Core i5-2450M, AMD Athlon X2 270, Intel Core i5-3470s; GPU: GTX550Ti, GT610, GT 520MX, Quadro K5000, Tesla K20), and three different software models (sequential, OpenMP and CUDA). Nine benchmark functions are selected as test problems, and average execution time is taken as a performance criterion. The results show that:
a) parallel implementations are faster than sequential codes as expected,
b) application oriented hardware might demonstrate unexpectedly the worst performance;
c) in general, GPU implementations outperform to CPU implementations, and
d) instead of mid-level GPU devices, the programmers can select CPUs as target devices, which prove to be more cost-effective.

CHAPTER 3

SYSTEM DESIGN

Parallelization of Particle Swarm Optimization (PPSO) can be briefly summarized as follows:

1. The position and velocity values of each particle are assigned randomly at different cores.
2. At the beginning of the iteration, the cost function is evaluated (f(xi)) for each position (xi) at each core
3. Personal (the best position of each individual between the iteration 1 and k) and global best (the best position among the population at the iteration k) positions are determined according to the cost values. This step includes calculation of taking the minimum of a vector, which is implemented by association of many cores.
4. The position of each particle is altered by using (1) and (2) at each core.
5. If the maximum number of iteration is not equal to current iteration k or minimum error criteria is not attained, then step b is revisited; else the algorithm is terminated. Figure 1 presents a good description for explaining PPSO

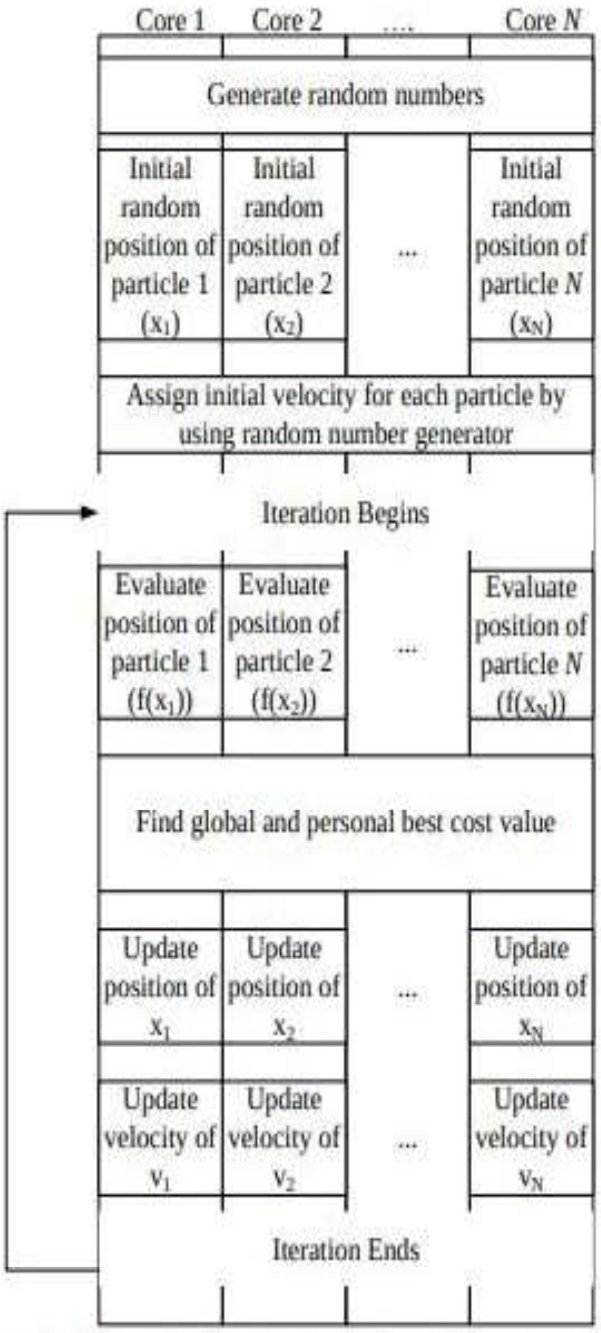


Figure 1. A sample demonstration for parallel particle swarm optimization

CHAPTER 4

IMPLEMENTATION

4.1 MPI Implementation

In this project, Message Passing Interface (MPI) based parallel computation and particle swarm optimization (PSO) algorithm are combined to form the parallel particle swarm optimization (PPSO) method for solving the minimisation or maximisation problems. In this method, the optimization process is implemented on a multi-processor supercomputer to reduce the computational time drastically.

There are a few approaches to apply parallel computation for PSO, such as the shared-memory approach and the message passing interface approach, etc. The MPI approach is chosen because of following advantages:

- MPI is probably the world's most widely-supported communications library for high-performance computing.
- With MPI's approximately 125 functions, programmers needn't to implement common communication structures themselves.
- It is easy to learn MPI, as a complete message passing program can be written with only six basic functions: MPI_Init, MPI_Finalize, MPI_Comm_rank, MPI_Comm_size, MPI_Send, and MPI_Recv.
- MPI has portability to almost all the major platforms.

MPI based parallel computation using n processors(how it works):

1. First, processor 0 reads and preprocesses all the data before scattering them to other processors' memory.
2. After processor 0's initialization work, all the processors calculate with the data that are sent by processor 0 through message passing interface.
3. When the calculations are done, results are sent back to processor 0's receiving buffer to be dealt with. Then processor 0 decides whether to continue computing or output final results and terminate program

The optimization program flow can be described as follows (Assume there are n processors running this program simultaneously. Processor 0 is the master node. The others are slave nodes.):

1. Processor 0 - processor n − 1 initialize PPSO parameters and MPI environment.

2. Let the time interval counter(time stamp) nt = 1.

3. Processor 0 randomly generates a swarm of particles, each of which stands for a candidate solution to minimisation or maximisation problem. Let the size of particle population be p.

4. Let m = p / n, m is an integer. If n can't be divided exactly, let m = m + 1, thus m ∗ n > p. Processor 0 partitions the swarm to n parts, each of which has m or m − 1 particles. Then processor 0 sends these parts to other processors using MPI_Bcast() function. Each processor receives m or m − 1 particles from processor 0 using the same MPI_Bcast() function.

5. Each processor evaluates the fitness of its own particles based on the particular function.

6. Processor 0 gathers the fitness values of all particles from processor 1-processor n − 1 using the MPI function MPI_Gather().

7. Processor 0 executes the PSO operator and adjusts all particles' positions in the search hyperspace.

8. Processor 0 outputs the best solution of the current time interval. If the time interval counter nt < Nt , let nt = nt + 1, (Nt = maximum iteration)

9. Processor 0 sends the global best solution to all the other processes using the MPI function MPI_Bcast(). This value is received by the other processes for the calculations in the further iterations

10. Go to step 3 if the stopping criteria is reached. Otherwise, go to step 10.

11. All processors exit MPI environment and terminate the program.

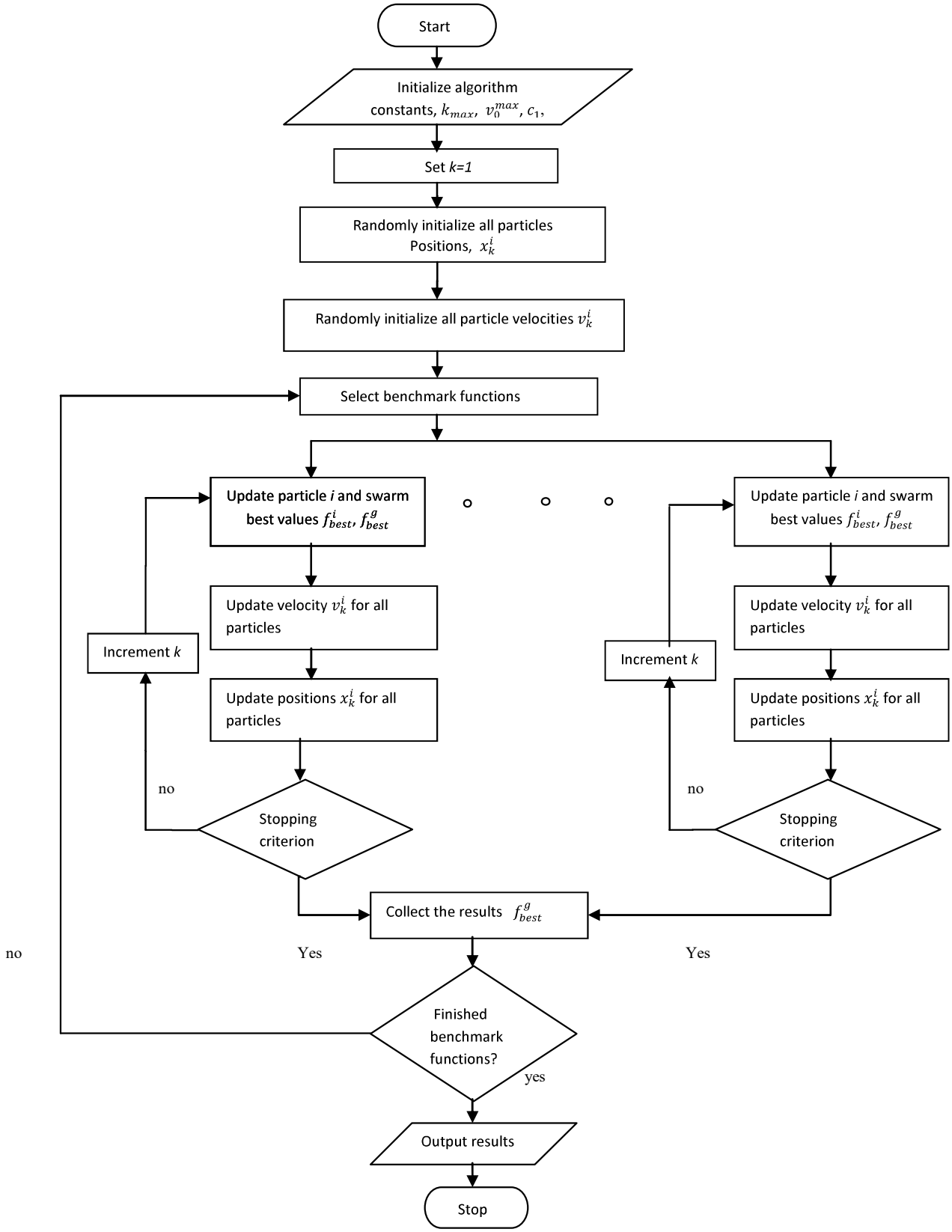


Figure 2: Flowchart for Parallel PSO

4.2 OpenMP Implementation

In a general manner, the codes which are written in C/C++ environment are executed on CPU on a single core. Since the numbers of the cores in CPU have increased, programmers desire to use these cores with simple modifications on sequential codes. Therefore, the OpenMP package was introduced for this purpose. This package contains compiler directives, run time routines and environment variables. Since all data is recorded on the globally shared memory on the CPU, OpenMP programming is much simpler. PPSO is also implemented on different CPU architectures.

1. All the particles are initialized with the initial parameters using the omp for construct. Global Best value is calculated reduction clause.

2. In each iteration, calculation for each particle is shared between different threads using the omp for construct. Number of threads to be used is predefined.

3. Each local best is considered the the global best position among them is calculated sequentially for proper synchronization.
4. These steps repeated until we reach the most optimal value or the maximum number of iterations, whichever is reached first.

4.3 MPI - OpenMP Hybrid

In MPI implementation each processor takes the calculcations responsible for only a certain amount of particles. This is run serially. This can be further parallelized by making these calculations under each processors parallel. OpenMP constructs are used for achieving this goal.

1. Processor 0 - processor n − 1 initialize PPSO parameters and MPI environment.

2. Let the time interval counter(time stamp) nt = 1.

3. Processor 0 randomly generates a swarm of particles, each of which stands for a candidate solution to minimisation or maximisation problem. Let the size of particle population be p.

4. Let m = p / n, m is an integer. If n can't be divided exactly, let m = m + 1, thus m ∗ n > p. Processor 0 partitions the swarm to n parts, each of which has m or m − 1 particles. Then processor 0 sends these parts to other processors using MPI_Bcast() function. Each processor receives m or m − 1 particles from processor 0 using the same MPI_Bcast() function.

5. Each processor evaluates the fitness of its own particles based on the particular function using OpenMP for construct.

6. Processor 0 gathers the fitness values of all particles from processor 1-processor n − 1 using the MPI function MPI_Gather().

7. Processor 0 executes the PSO operator and adjusts all particles' positions in the search hyperspace.

8. Processor 0 outputs the best solution of the current time interval. If the time interval counter nt < Nt , let nt = nt + 1, (Nt = maximum iteration)

9. Processor 0 sends the global best solution to all the other processes using the MPI function MPI_Bcast(). This value is received by the other processes for the calculations in the further iterations

10. Go to step 3 if the stopping criteria is reached. Otherwise, go to step 10.

11. All processors exit MPI environment and terminate the program.

4.4 Testing

The numerical problems may be divided into two classes; benchmark problems and real life problems. For the present study we have taken three benchmark functions to analyze the behavior of our PSO algorithm:

1)Rosenbrock's function

$$f(x, y) = (a - x)^2 + b(y - x^2)^2$$

whose global minimum f = 0 . It can be treated as a multimodal problem. It has a narrow valley from the perceived local optima to the global optimum.
Search range: [-2.48, 2.48]

2) Ackley Function

$$f(x, y) = -20\exp\left[-0.2\sqrt{0.5\left(x^2 + y^2\right)}\right] - \exp[0.5\left(\cos 2\pi x + \cos 2\pi y\right)] + e + 20$$

This function has the global minimum f = 0 at x = (0, 0, ..., 0).Ackley's function has one narrow global optimum basin and many minor local optima.
It is probably the easiest problem among the six as its local optima are neither deep nor wide.
Search range: [-32.76, 32.76]

3)Schwefel's function

$$f(\mathbf{x}) = 418.9829d - \sum_{i=1}^{d} x_i \sin(\sqrt{|x_i|})$$

Schwefel's function is due to its deep local optima being far from the global optimum. It will be hard to find the global optimum, if many particles fall into one of the deep local optima.Schewefel function is multimodal functions having several optima.It has global minimum f=0 .
Search range: [-5.12, 5.12]

CHAPTER 5
RESULTS

The first step is to examine the time taken to run the original PSO code on a single core without any parallelization and then compare the result of the experiment with the result of another experiment that was conducted using the parallel PSO code. In both the experiments all benchmark functions discussed above are evaluated. Two scaling experiments were conducted to see the parallelization effect in terms of speedup based on
i) Number of iterations
ii)Number of particles

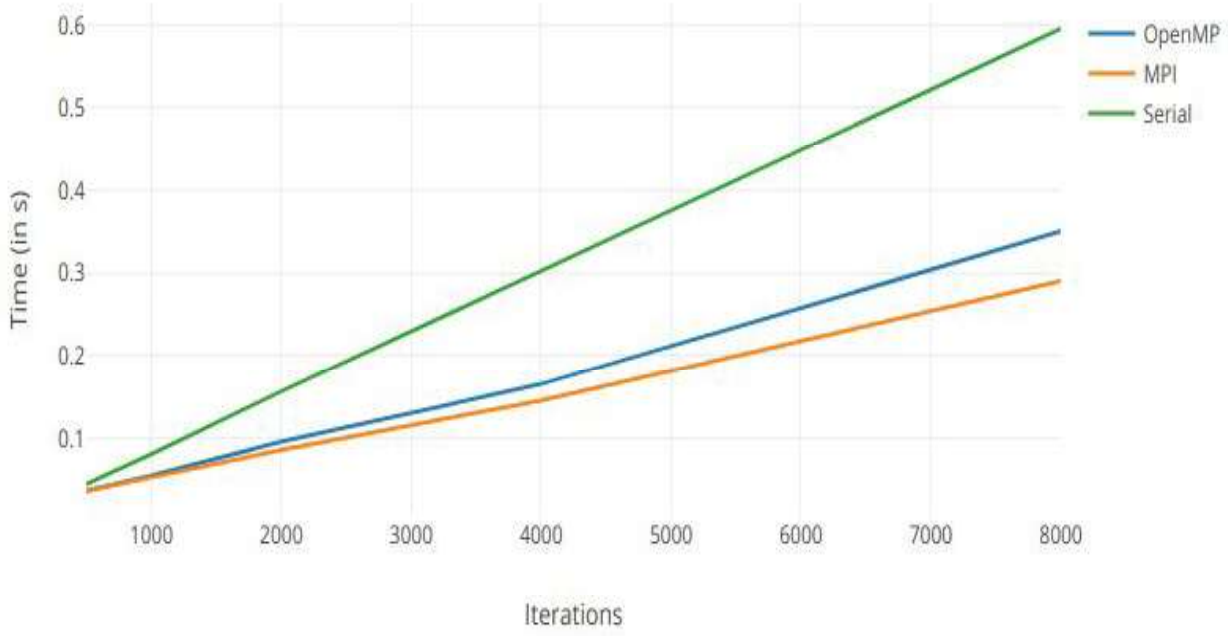
5.1 Scaling Experiment 1:

In this experiment, serial and parallel PSO is made to run with constant number of particles(= 100) and constant number of dimensions(= 5) and and varying number of maximum iterations.
The below table gives the result of experiment:
Following inferences can be made:

1) Initially,the time execution of all codes are similar and then as number of iterations increases the parallelised code has achieved speedup upto 100%. This can be attributed to parallel overhead for smaller no. of iterations which becomes negligible when no. of iterations increase.
2) In both OpenMp and MPI, the time execution reduces as no of threads/process are increased with least time at four threads/processes i.e. due to increase in parallelisation.
3) MPI takes lesser time than OpenMP.

Number of particles = 100
Number of dimensions = 5

		OpenMP (Time in s)			MPI		Serial
Iterations	2	3	4	2	3	4	
500	0.032	0.029	0.036	0.031	0.030	0.036	0.044
1000	0.058	0.054	0.054	0.058	0.052	0.052	0.08
2000	0.11	0.100	0.095	0.095	0.09	0.085	0.156
4000	0.205	0.196	0.165	0.175	0.170	0.145	0.303
8000	0.405	0.385	0.35	0.35	0.33	0.29	0.595



5.2 Scaling Experiment 2:

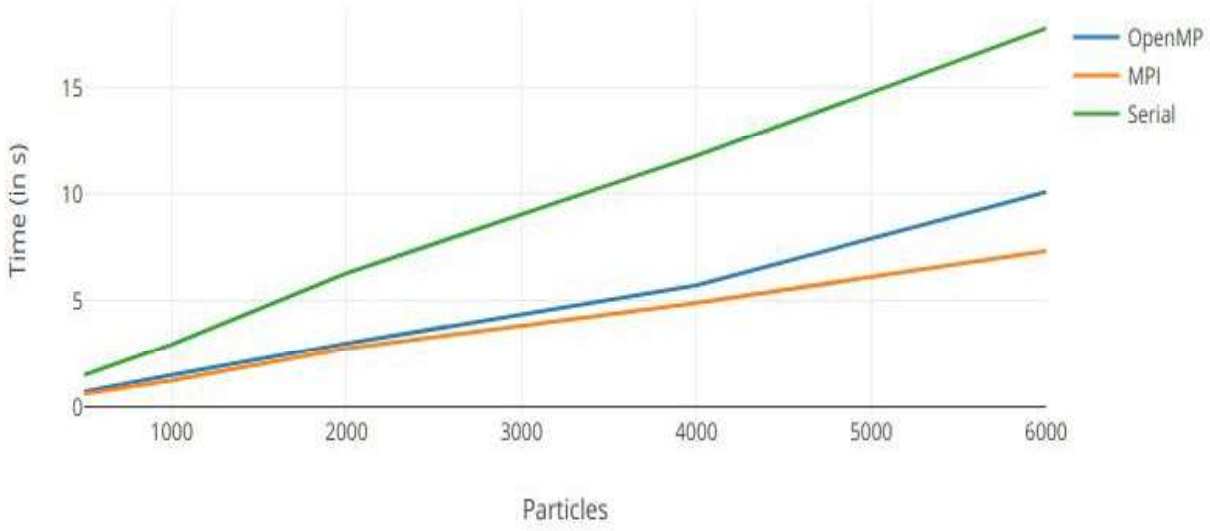
In this experiment, serial and parallel PSO is made to run with constant number of iterations (2000) and constant number of dimensions (10) and and varying number of maximum iterations. The below table gives the result of experiment:
Following inferences can be made:

1. MPI and OpenMp achieve more speedup up than the last experiment with MPI speeding upto three times the serial code.This is because the parallelisation is done primarily on particles.
2. In both OpenMp and MPI, the time execution reduces as no of threads/process are increased with least time at four threads/processes i.e. due to increase in parallelisation.Also there is drastic change between time execution of two threads/process and four threads/process,hence number of parallelisation directly affects time execution.
3. MPI takes lesser time than OpenMP.

Number of iterations = 2000

Number of dimensions = 10

	OpenMP (Time in s)			MPI			Serial
Particles	2	3	4	2	3	4	
500	1.15	0.90	0.73	0.81	0.75	0.65	1.5
1000	2.22	1.76	1.52	1.82	1.50	1.25	2.96
2000	4.54	3.65	3.01	3.17	3.08	2.76	6.31
4000	9.23	7.19	5.75	6.42	6.07	4.91	11.81
6000	13.86	11.22	10.10	9.67	9.53	7.35	17.82



CHAPTER 6
CONCLUSION

Here we can conclude that parallel program execution is more efficient than serial execution when the number of particles and number of iterations are very high. We can observe from this project that Parallelization has more effect when on particles is more than the on iteration.
In this project we have also observed that MPI execution is slightly better than OpenMP execution.

6.1 CODE

https://github.com/abhilashvenkatesh/Parallelization-of-PSO/

6.2 REFERENCES

[1]:https://www.sciencedirect.com/science/article/pii/S0898122108005403

[2]:https://library.ndsu.edu/ir/bitstream/handle/10365/25649/Parallel%20Particle%20Swarm%20Optimization.pdf?sequence=1&isAllowed=y

[3]:http://isites.info/PastConferences/ISITES2014/ISITES2014/papers/B6-ISITES2014ID323.pdf