SPM Project 2019
Using *Particle swarm optimization* to find minimum of a function.

Yanamandra Sharath Chandra (585953)

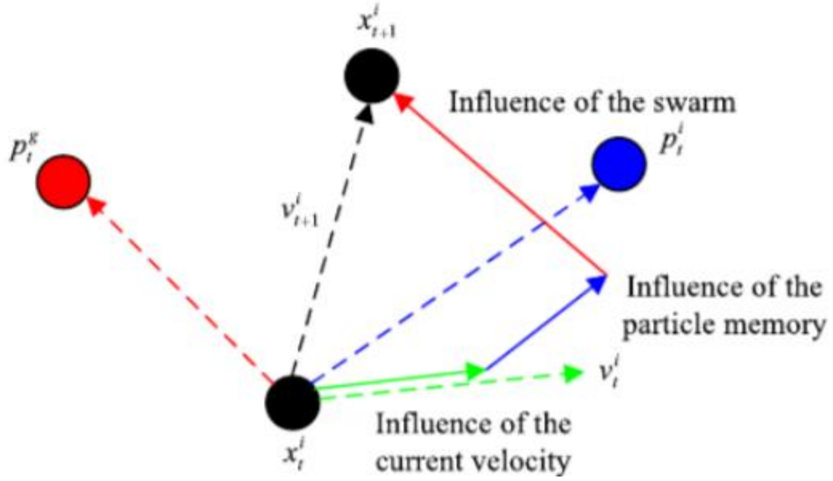
University of Pisa

Aim of the Project:

To develop an application finding the minimum of an object function in each interval using swarm particle optimization. The Swarm particle Optimization uses particles to explore the state space of solution. It is an iterative process where a set of particles, initially spread randomly across the solution space, and autonomously travel in the space driven by local and global knowledge relative to the solution explored so far.
In this report on *Swarm Particle Optimization (SPO)*, I analyzed the working, implementation and principle both theoretically and practically in three various versions, *sequential*, *multi thread (without library)* and using *Fastflow library.*

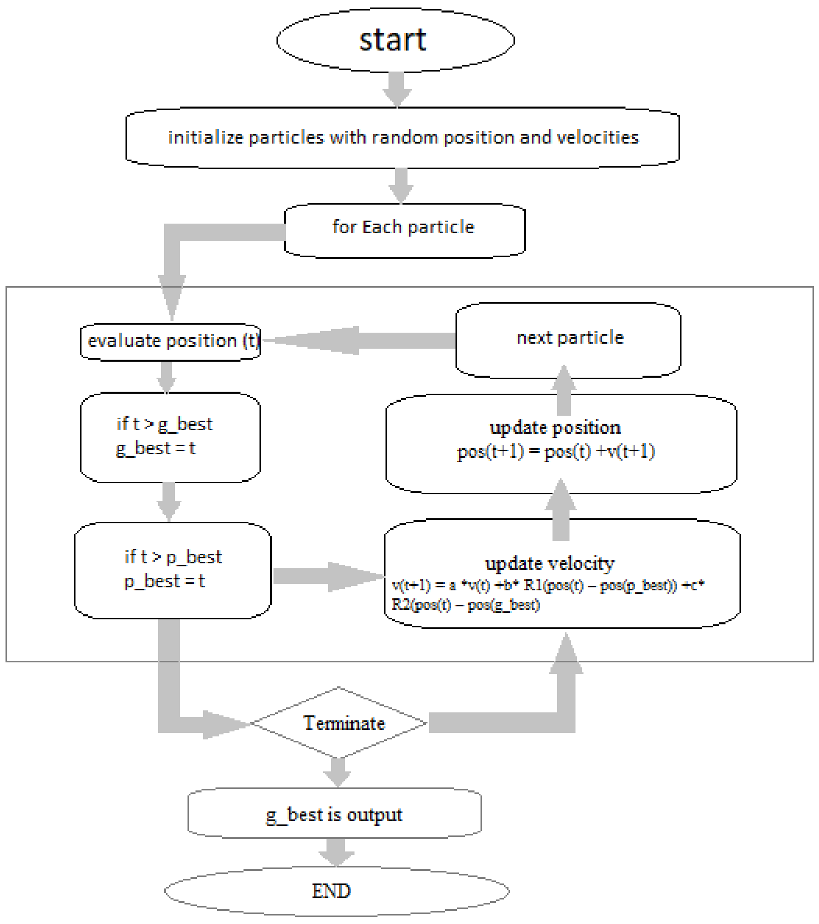
Introduction:

Swarm particle optimization is population based stochastic optimization technique given by Dr. Eberhart and Dr. Kennedy in 1995. It is a meta heuristics approach which is inspired by social behavior of nature. It has been applied in: Artificial neural network training, fuzzy control systems and many areas of genetic algorithm.



Working of SPO, Initially the particle is driven by three components: vector velocity, particle best position (p_best) and global best (g_best). And updated its position and velocity of individual particles. The SPO works on communication and learning of particles by its experience and generation.

Design architecture:



Approach:

The general principle of SPO categorized into phases, but implementation varies according to versions taken:

1. Initialization: The algorithm initializes a population of random solutions and searches for global optima by updating its optima of generations. In this phase the particles are distributed across the space and computed local optima and the best local optimum is declared as best global optimum till then. And randomly initialize velocities to the particles.

2. Iteration: In this iteration phase the position of each particles with current velocity is updated till the iteration are finished.
   And in addition if the particle moves away from search space we assign it back to its minimum and reevaluate local and global optimum.

3. Update position and velocity:

   To **update velocity** of particle:
   v(t+1) = w *v(t) +c1* R1(pos(t) – pos(local_best)) +c2* R2(pos(t) – pos(global_best))

To **update position** of particle:

$$pos(t+1) = pos(t) + v(t+1)$$

Implementation:

Task: Using Particle swarm optimization finding the minimum of the function.

Approach: Aim of this project is to tackle the problem in three different approaches, sequential, parallel multi thread and FASTFLOW library.

Requirements/ Machine configuration: Intel(R) core i5 – 8250U CPU @ 1.6GHz 1.80GHz, 16gb RAM with 4 cores 8 logical cores.

Remote machine: Intel Xeon Phi KNL, with a CPU with 64 cores (256 threads), each @ 1.30 GHz.

Performance modeling for Sequential, Multi thread without using library and using FastFlow library approaches are given below with detailed explanation along with graphs.

Total number of particles taken = 10000 particles
Number of iterations = 500 iterations
Objective function = x + y (To find the minimum)

Run the Code:

Library Path:
```
LD_LIBRARY_PATH=/usr/local/lib64/:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH
```

Sequential Version commands:
```
g++ sequential.cpp -o sequential -O3
. / sequential
```

Multi thread Version commands:
```
g++ nthread.cpp -pthread -O3
. / a.out n
```
n is number of workers (1 ,2, 4, 8, 16, 32, 64, 128, 256)

Fastflow Version commands:
```
g++ fastflow_2.cpp -pthread -std=c++17 -I/home/sharath/desktop/pds_19/ff   -O3
. / a.out n
```
n is number of workers (1 ,2, 4, 8, 16, 32, 64, 128, 256)

We need to provide Fastflow library path while compiling with O3 flag.

Compilation time

Sequential

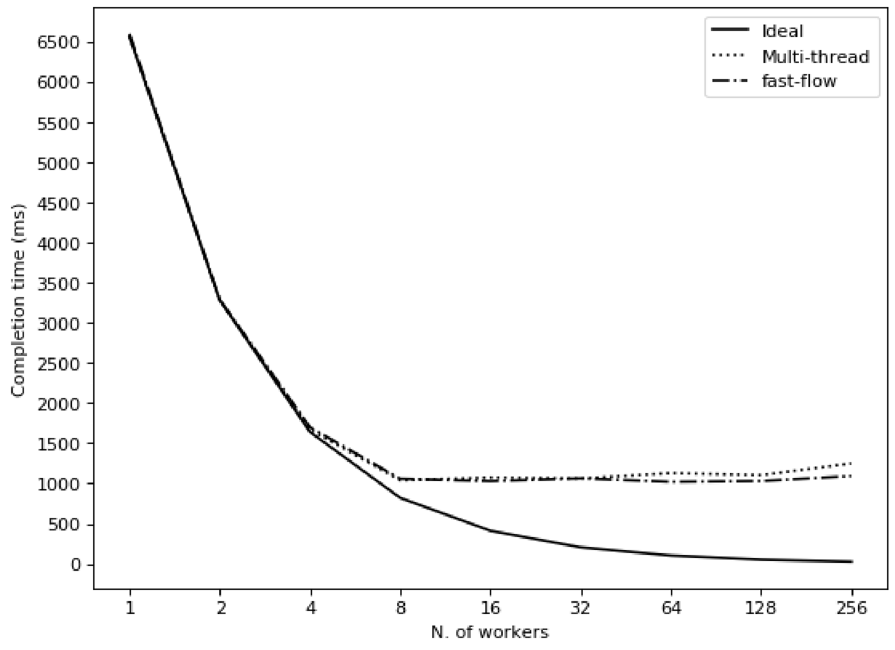
	Local machine	Remote machine
	In milliseconds	In milliseconds
Completion Time	6562	13233

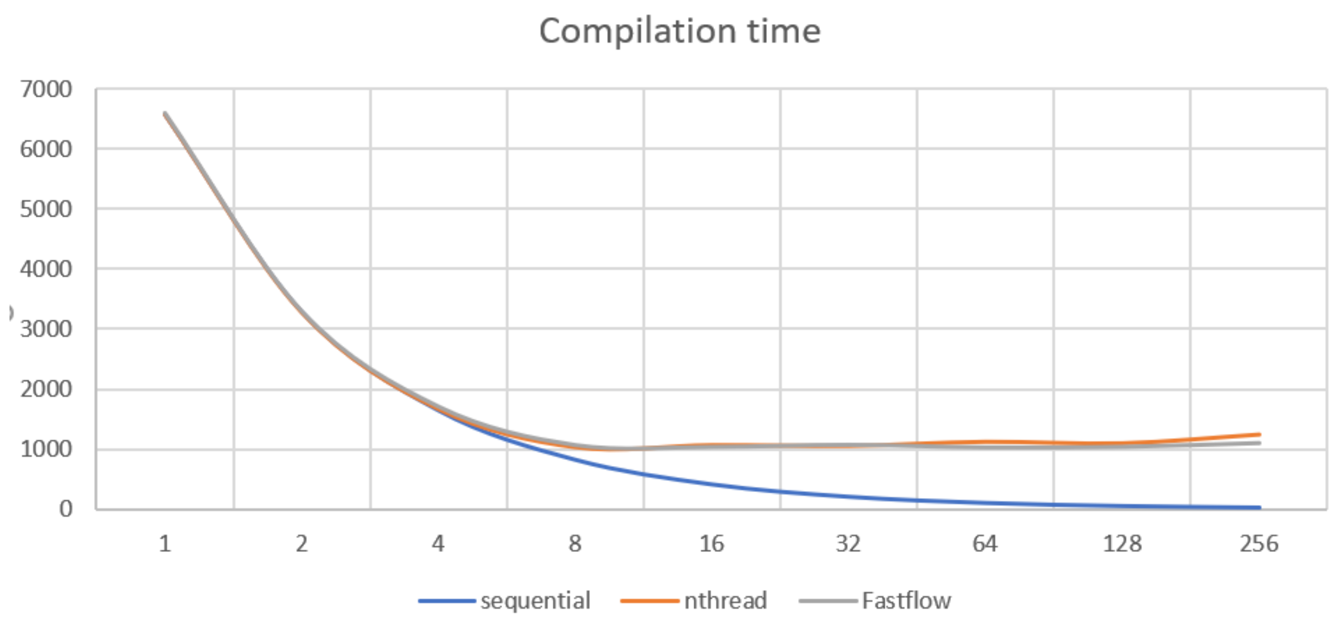
Ideal compilation time is calculated by:
Ideal time = sequential time / number of workers

Local machine

No: of threads	1	2	4	8	16	32	64	128	256
nThread	6564	3278	1671	1039	1074	1061	1129	1104	1249
FastFlow	6598	3292	1700	1058	1029	1063	1020	1030	1092

Completion time on local machine





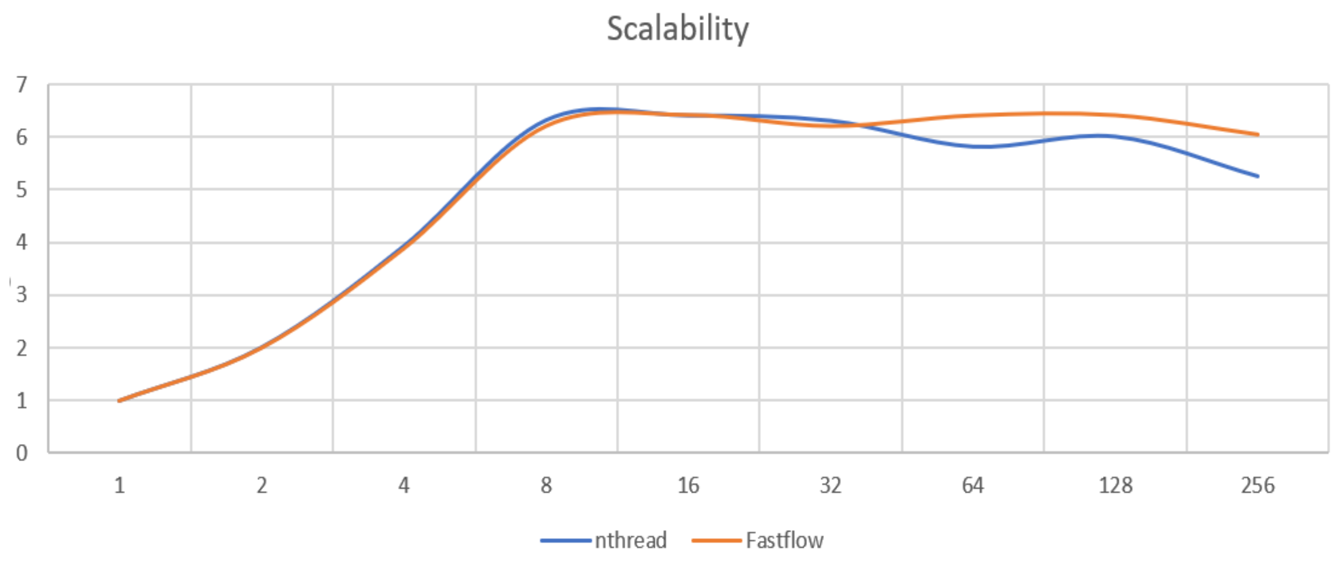
Performance evaluation on Local machine

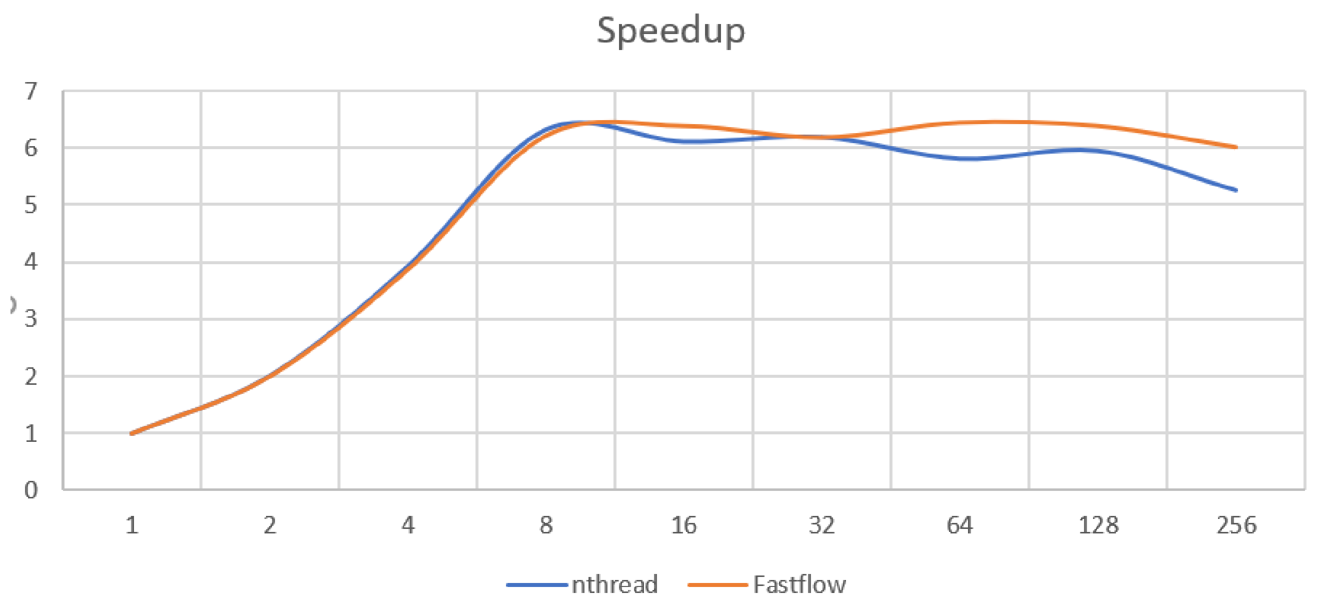
Multi thread (nthread)

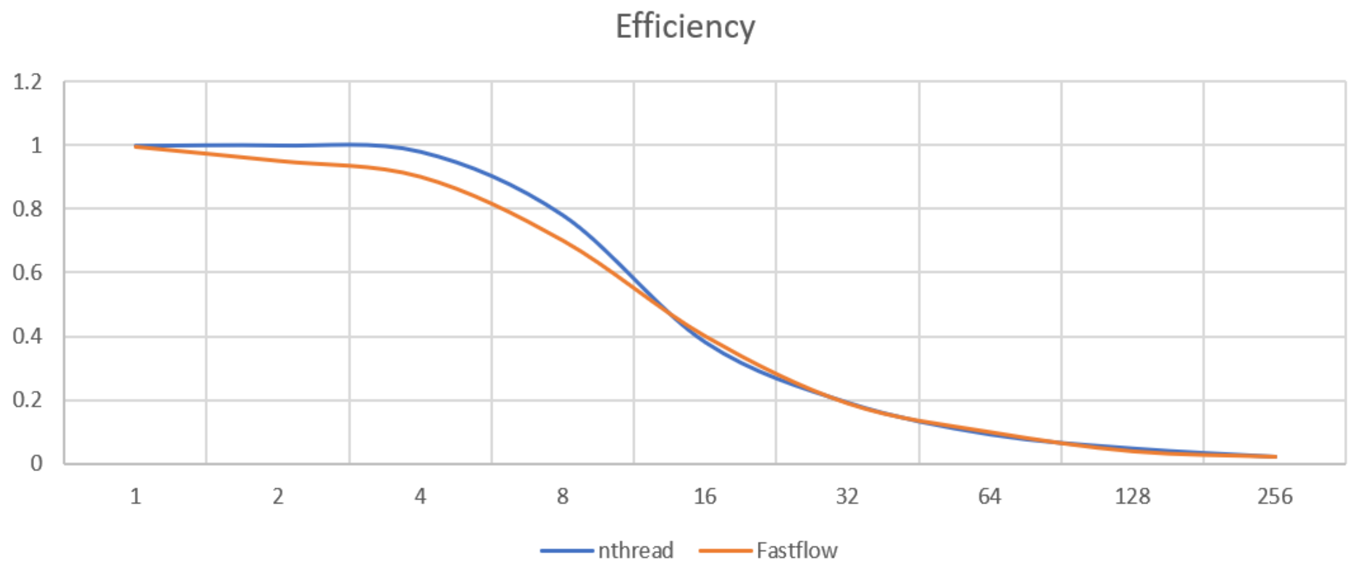
Number of workers	Scalability Tpar(1)/Tpar(nw)	Speedup Tseq/Tpar(nw)	Efficiency speedup / nw
1	1	0.99	0.998
2	2.00	2.00	1
4	3.92	3.92	0.98
8	6.31	6.31	0.78
16	6.40	6.10	0.38
32	6.18	6.18	0.19
64	5.81	5.8	0.09
128	6.00	5.94	0.046
256	5.25	5.25	0.020

Fastflow (parallel_for)

Number of workers	Scalability Tpar(1)/Tpar(nw)	Speedup Tseq/Tpar(nw)	Efficiency speedup / nw
1	1	0.994	0.994
2	2.00	1.993	0.95
4	3.88	3.86	0.90
8	6.2	6.20	0.77
16	6.41	6.37	0.39
32	6.2	6.17	0.19
64	6.4	6.43	0.10
128	6.405	6.37	0.049
256	6.04	6.00	0.023


Scalability
7
6
5
4
3
2
1
0
1
2
4
8
16
32
64
128
256
nthread
Fastflow

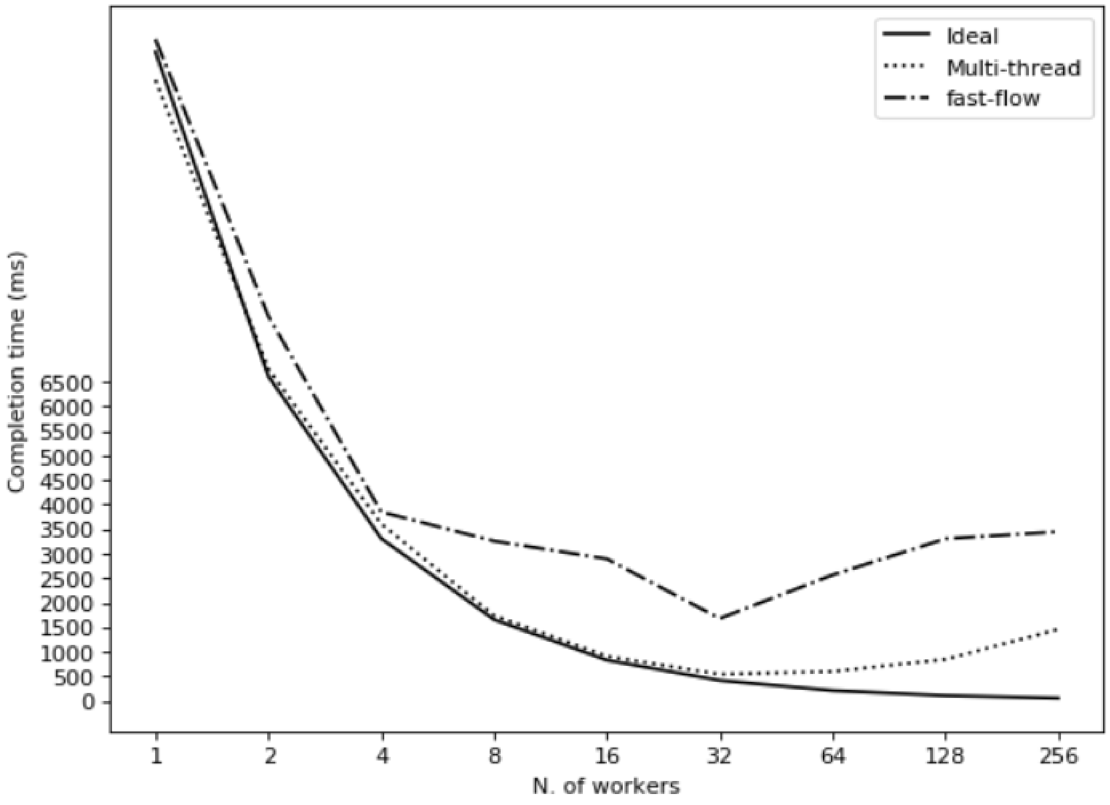

Speedup
7
6
5
4
3
2
1
0
1
2
4
8
16
32
64
128
256
nthread
Fastflow

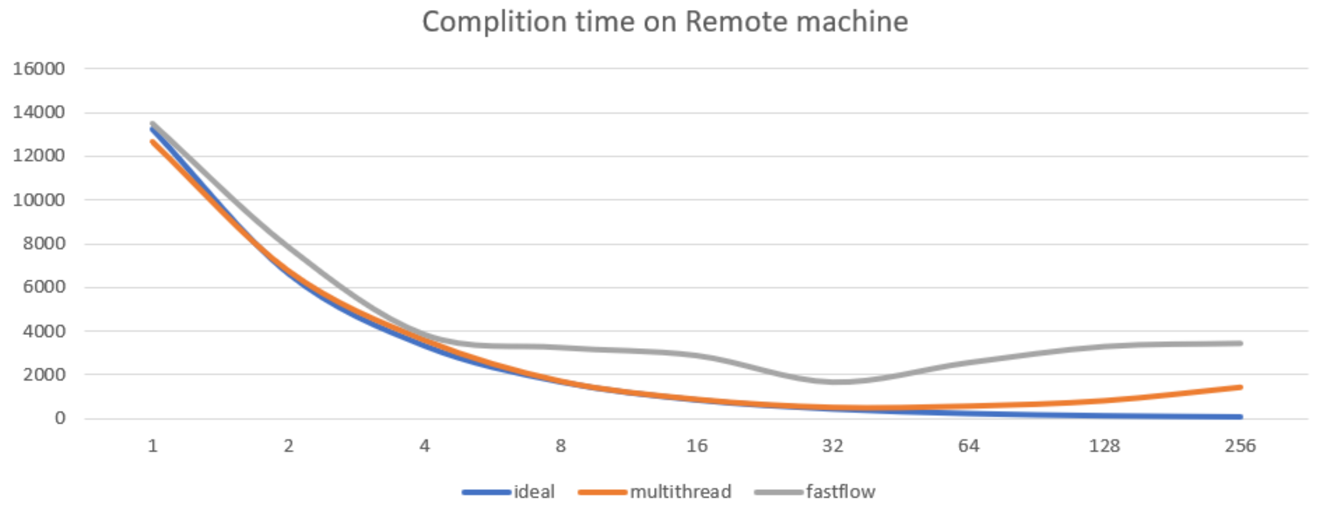

Efficiency
1.2
1
0.8
0.6
0.4
0.2
0
1
2
4
8
16
32
64
128
256
nthread
Fastflow

Remote machine

No: of threads	1	2	4	8	16	32	64	128	256
nThread	12657	6765	3608	1735	906	539	596	843	1456
FastFlow	13486	7843	3850	3256	2892	1676	2565	3303	3447

completion time on Remote machine





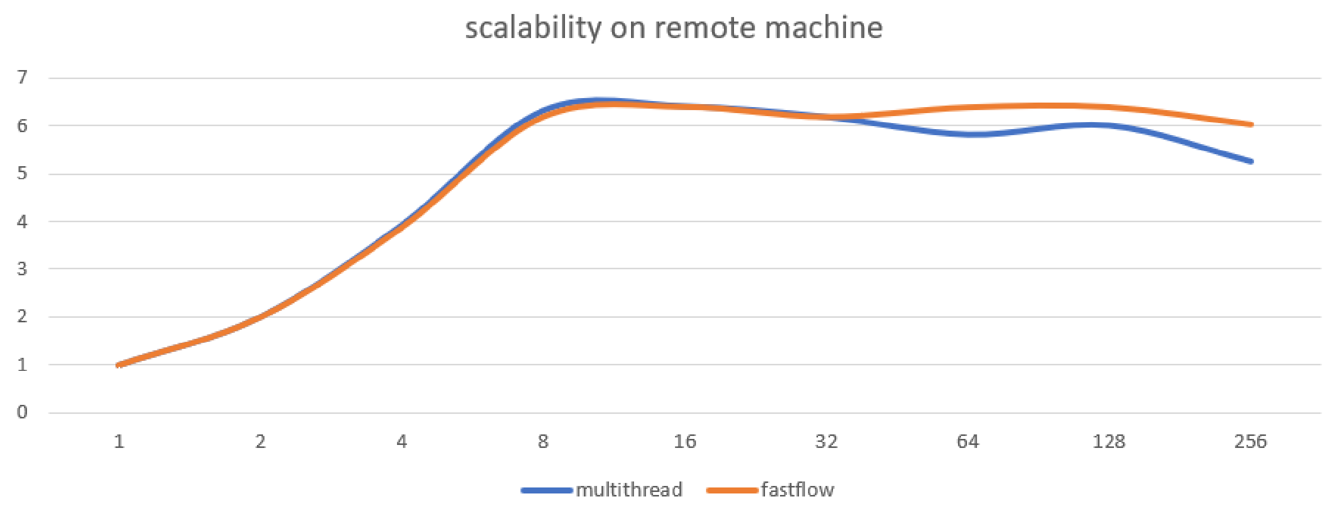
Performance evaluation on Remote machine

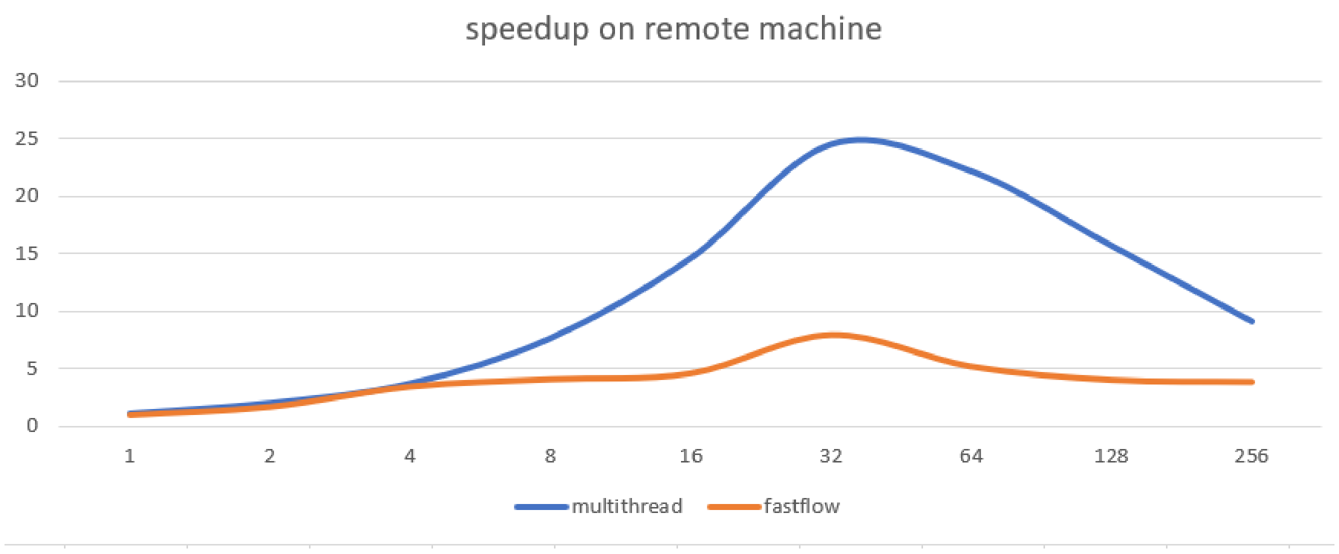
Multi thread(nthread)

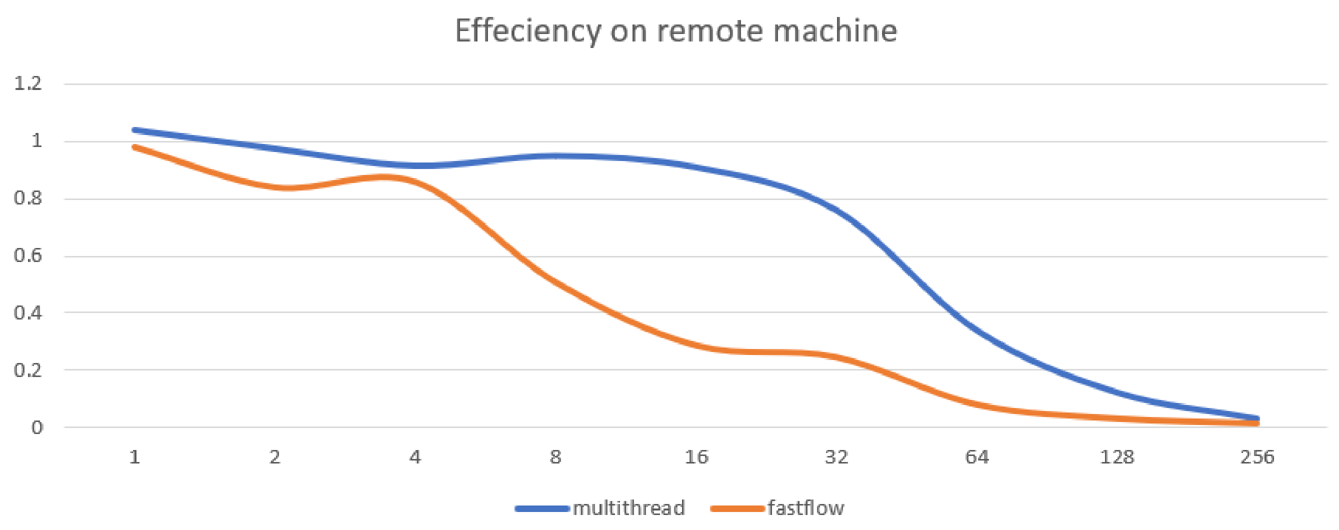
Number of workers	Scalability Tpar(1)/Tpar(nw)	Speedup Tseq/Tpar(nw)	Efficiency speedup / nw
1	1	1.04	1.04
2	1.87	1.95	0.975
4	3.50	3.66	0.915
8	7.29	7.62	0.95
16	13.97	14.60	0.91
32	23.48	24.55	0.76
64	21.23	22.20	0.34
128	15.01	15.69	0.122
256	8.69	9.08	0.03

Fastflow (parallel_for)

Number of workers	Scalability Tpar(1)/Tpar(nw)	Speedup Tseq/Tpar(nw)	Efficiency speedup / nw
1	1	0.981	0.981
2	1.719	1.687	0.84
4	3.502	3.437	0.859
8	4.141	4.064	0.508
16	4.663	4.57	0.287
32	8.046	7.895	0.246
64	5.257	5.159	0.080
128	4.082	4.006	0.031
256	3.912	3.838	0.014







Key implementation in **fastflow** library which are used in this project is 'parallel_for ', helps to initialize the particles and performing iteration in parallel.

Conclusion

The minimum of an object function is determined by using 'Swarm Particle Optimization' in three versions (sequential, multi thread and Fast flow library).