Arabian Journal for Science and Engineering (2019) 44:2899–2923
https://doi.org/10.1007/s13369-018-03713-6

REVIEW - COMPUTER ENGINEERING AND COMPUTER SCIENCE

A Survey on Parallel Particle Swarm Optimization Algorithms

Soniya Lalwani[1] · Harish Sharma[1] · Suresh Chandra Satapathy[2] · Kusum Deep[3] · Jagdish Chand Bansal[4]

Received: 13 July 2018 / Accepted: 31 December 2018 / Published online: 8 January 2019

Abstract
Most of the complex research problems can be formulated as optimization problems. Emergence of big data technologies have also commenced the generation of complex optimization problems with large size. The high computational cost of these problems has rendered the development of optimization algorithms with parallelization. Particle swarm optimization (PSO) algorithm is one of the most popular swarm intelligence-based algorithm, which is enriched with robustness, simplicity and global search capabilities. However, one of the major hindrance with PSO is its susceptibility of getting entrapped in local optima and; alike other evolutionary algorithms the performance of PSO gets deteriorated as soon as the dimension of the problem increases. Hence, several efforts are made to enhance its performance that includes the parallelization of PSO. The basic architecture of PSO inherits a natural parallelism, and receptiveness of fast processing machines has made this task pretty convenient. Therefore, parallelized PSO (PPSO) has emerged as a well-accepted algorithm by the research community. Several studies have been performed on parallelizing PSO algorithm so far. Proposed work presents a comprehensive and systematic survey of the studies on PPSO algorithms and variants along with their parallelization strategies and applications.

Keywords Particle swarm optimization · Parallel computing · Swarm intelligence-based algorithm · GPU · MPI · Large-size complex optimization problems

Abbreviations

AWS	Amazon web services
CA	Cellular automata
CNOP	Conditional nonlinear optimal perturbation
CPU	Central processing unit
CUDA	Compute unified device architecture
DNN	Deep neural networks
DORPD	Dynamic optimal reactive power dispatch
ED	Economic dispatch
FJSP	Flexible job shop scheduling problem
FPGA	Field programmable gate array
GA	Genetic algorithm
GPU	Graphics processing unit
HPF	High-performance Fortran
HSI	Hyper spectral images
JSSP	Job shop scheduling problem
MOP	Multi-objective optimization problem
MPI	Message-passing interface
NMR	Nuclear magnetic resonance
OpenCL	Open computing language
OpenGL	Open graphics library
OpenMP	Open multiprocessing
PPSO	Parallel particle swarm optimization
PSO	Particle swarm optimization
PVM	Parallel virtual machine
QoS	Quality of service
SA	Simulated annealing

✉ Soniya Lalwani
slalwani.math@gmail.com

Harish Sharma
harish.sharma0107@gmail.com

Suresh Chandra Satapathy
suresh.satapathyfcs@kiit.ac.in

Kusum Deep
kusumdeep@gmail.com

Jagdish Chand Bansal
jcbansal@gmail.com

[1] Department of Computer Science and Engineering, Rajasthan Technical University, Kota, India

[2] School of Computer Engineering, Kalinga Institute of Industrial Technology, Bhubaneswar, Odisha, India

[3] Department of Mathematics, Indian Institute of Technology, Roorkee, India

[4] South Asian University, New Delhi, India

SMP	Symmetric multiprocessing
TPU	Tensor processing unit
TSVD	Truncated singular value decomposition
UAV	Unmanned aerial vehicle
V2G	Vehicle-to-grid

1 Introduction

Real-world optimization problems are usually complex, large-scale and NP-hard. They not only contain the terms of constraints, single/multiple objectives, but also their modeling gets continuously evolving. Their resolution and iterative evaluation of objective functions require long CPU time. PSO is a population-based metaheuristic that has proven itself to be one of the most efficient nature-inspired algorithms to deal with unconstrained and constrained global optimization problems with one or many objectives. But the global convergence is not assured with PSO algorithms due to the constraint of particles to stay in a finite sampling space. By virtue of this, it may bring premature convergence by diminishing the global search ability of the algorithm [1]. Hence, many strategies are being proposed to improve its' efficiency that includes the parallelization of PSO. Since PSO algorithms are population-based, they are intrinsically parallel. Hence, PPSO has become one of the most popular parallel metaheuristic [2].

Parallelization proposes an excellent path to enhance the system performance. For parallelization, multi-core CPU or GPU can be occupied. The noteworthy important issues in parallelization are the operating system, communication topologies, programming languages enriched with modules, functions and libraries. The parallelization options include: Hadoop MapReduce, CUDA, MATLAB parallel computing toolbox, R Parallel package, Julia: Parallel for and MapReduce, OpenCL, OpenGL, Parallel computing module in python, OpenMP with C++ and Rcpp, POSIX threads, MPI, HPF, PVM, and Java threads on SMP machines. Moreover, cloud computing services offer access to large servers containing several CPUs and GPUs for providing massively parallel programming [3,4]. A few of these services include: Amazon elastic compute cloud and Google cloud compute engine. Here, for the sake of ubiquity, the most popular parallelization strategy and communication models are discussed in next sections.

Present work is a chronological literature review of the available PPSO versions, collected from various internet sources. The PPSO versions include individual variants, application-based, parallelization strategy based and number of problem objectives based variants. Initially, the text is classified at the basis of CPU and GPU implementation.

Classification of the presented work is as follows: Sect. 2 presents the details of PSO and its parallelization strategies, communication models and few conventional parallel PSO algorithm variants; Sect. 3 is based on presenting the summary of the studies performed on PPSO so far, which is classified at the basis of CPU- and GPU-based implementation. Finally, a comparative analysis on the basis of parallel computing models and purpose of using PPSO is performed in Sect. 4 along with the conclusion of the presented work in Sect. 5.

2 Parallel Particle Swarm Optimization: An Overview

2.1 Particle Swarm Optimization Algorithm

PSO algorithm was derived by Kennedy and Eberhart in 1995 for simulating the behavior of a bird flock or fish school [2]. They move in different directions while communicating to each other, updating their positions and velocities for the better position that may contribute toward optimal solution. The objective function to be minimized (/maximized) is formulated as:

$$\min f(x) \quad s.t. \quad x \in S \subseteq R^D \tag{1}$$

where x is decision variable matrix, comprised of m vectors with dimension D, defined as $x = [\overrightarrow{x}^1, \overrightarrow{x}^2 \ldots \overrightarrow{x}^m]$ in feasible solution space S [5]. Previous velocity $v^i(t)$ and position $x^i(t)$ are updated by:

$$\begin{aligned} v^i(t+1) &= wv^i(t) + c_1 r_1[pbest^i(t) - x^i(t)] \\ &\quad + c_2 r_2[gbest(t) - x^i(t)] \end{aligned} \tag{2}$$

$$x^i(t+1) = x^i(t) + v^i(t+1) \text{ with } x^i(0) \in \mathrm{U}(x_{\min}, x_{\max}) \tag{3}$$

The velocity v^i lies between lower and upper bound, i.e., $[v_{\min}, v_{\max}]$, where $v_{\min} = -v_{\max}$; w is inertia weight lying between 0 and 1, the scaling factor over the previous velocity; c_1 and c_2 are cognitive and social acceleration coefficients, respectively; r_1 and r_2 are uniform random numbers in range [0 1]. Particles personal best $pbest$ at iteration $(t+1)$ and best of the positions, i.e., $gbest(t)$ are updated as follows:

$$pbest^i(t+1) = \begin{cases} pbest^i(t) \text{ if } f(x^i(t+1)) \geq f(pbest^i(t)) \\ x^i(t+1) \text{ if } f(x^i(t+1)) < f(pbest^i(t)) \end{cases} \tag{4}$$

$$\begin{aligned} gbest(t+1) = x_k \in \{&pbest^1(t+1), pbest^2(t+1), \ldots \\ &pbest^m(t+1)\} \\ \text{where } f(x_k) = min\{&f(pbest^1(t+1)), f(pbest^2(t+1)), \\ &\ldots f(pbest^m(t+1))\} \end{aligned} \tag{5}$$

Two kinds of parallelisms are implemented in PSO to build PPSO algorithms: data parallelism and task parallelism. In data parallelism, same function/task is simultaneously executed on multiple processors/cores with distribution of the elements of a data set. Whereas, task parallelism is simultaneous execution of many different functions/tasks on multiple processors/cores for the same (or different) data sets. The parallelization depends upon the user's choice, i.e., whether the datasize is large or there are multiple number of tasks. In parallel version of PSO algorithms, different data sets in form of particles can be processed over multiple processors. Also, single data set could be taken for multi-objective or multi-task problems on the different processors with individual PSO algorithm implementations. Moreover, the implementation of PPSO could be in both the ways: synchronous and asynchronous. If every particle performs function evaluation in parallel and maintains synchronism with each other for all the iterations, the PPSO algorithm remains synchronous. Hence, the velocity and position of each particle is updated by the end of every iteration, whereas, in asynchronous PPSO algorithm, the particles do not synchronize with each other. Hence, position and velocity are updated continuously, based upon the available information. The details of parallel PSO are provided in Sect. 2.4.

2.2 Parallelization Strategies

Parallel computing is the simultaneous use of multiple computing resources to solve a computational problem by breaking it into discrete parts. This computation may occur on a single machine as well as on multiple machines. Single machine processing includes computers utilizing multi-core, multiprocessor, and GPU with multiple processing elements. Multiple machine examples include clusters, grids, and clouds [6]. Further, the parallelization strategies can be classified on the basis of their implementation platform [7] as described below:

2.2.1 CPU-Based Parallelization Strategies

These strategies take the advantage of accessing the multiple cores that may be physical or virtual with one or more CPUs. These approaches include:

- *Hadoop MapReduce*
  MapReduce programming model was established by Google for processing of large-sized data, which entails parallelization on each CPU (or single CPU) with distribution of different data. The implementation performs the operations of data distribution, parallelization, load balancing and fault tolerance as an inside process, whereas the user needs to perform only straightforward operations. Mapper requires an input pair and yields intermediate key/value pairs. Then, all the intermediate values containing the same intermediate key are grouped by the MapReduce library and sent to the reducer. The reducer then combines the corresponding values associated with the intermediate key to merge the set of values [8].
- *MATLAB parallel computing toolbox*
  MATLAB has provided the easiest parallel programming avenue by producing parallel computing toolbox. It is genuinely user-friendly, but incurs high cost upon purchase. In the parallel pool, the variables are accessible by any worker, so the basic task remains to initialize the parallel pool of workers and mentioning the 'for loop' that is required to be parallelized [9].
- *R Parallel package*
  R is an excellent open-source language with statistical and graphics competence. Developers have designed several parallel computing packages in R, out of which 'foreach' and 'doParallel' are extensively employed. Besides, C++ codes are embedded in R for executing parallel computing resulting Rcpp package [10].
- *Julia: Parallel for and MapReduce*
  Julia is an open-source programming language which is modern, functional, and expressive, and has remarkable abstraction and metaprogramming capabilities. Julia was designed with the aim of contributing toward powerful parallelization. If every parallel iteration requires few evaluations, then '@parallel for' from Julia is most suitable. It is basically created for the assignment of small tasks to each worker. But, for several control variables or for the models with multiple discrete choices '@parallel for' reduces the speed. Then, MapReduce function in Julia can be an excellent approach. It accepts inputs in form of function and vector of values for evaluating that function [11].
- *Parallel Computing module in Python*
  Python is a flexible open-source, interpreted, general-purpose language containing multiple modules. Out of which, Parallel function, a map-like function from the Joblib module, is very popular. In the parallel pool, all the declared variables are globally observable and modifiable by the workers [12].
- *OpenMP with C++*
  C++ is a compiled language with remarkably excellent speed, enriched with robustness and flexibility [13]. OpenMP is one of the simplest tools in C++ to perform parallel computing on multi-core/multiprocessor shared memory systems [14].
- *MPI*
  The parallel processes executed on distributed systems including multi-core/many-core processors perform communication *via* MPI. MPI is a library with a set of function calls and portable codes for supporting the performance optimization [15].

2.2.2 GPU-Based Parallelization Strategies

Last ten years have witnessed the increasing popularity of GPU-based parallelization. GPU has thousands of cores installed and strength of multiple CPUs in one processor. Any CPU-based parallelization strategy can be implemented in GPU as well. The most popular GPU-based parallelization schemes are:

- *CUDA*
  CUDA is a parallel computing model that allows parallel computing pursuit on the GPU of the computer for C, C++, Fortran, and Python. In November 2006, general-purpose parallel computing architecture called CUDA™ was introduced by nVIDIA™ [16], which is suitable for massively parallel computation too. To develop the parallel programs, a compatible GPU and the CUDA™ SDK are sufficient. User firstly needs to define the functions that are required to be run on the GPU followed by the memory allocation of the variables. Then, the process begins, starting from the initialization [17].
- *OpenACC*
  OpenACC has the analogous architecture to that of OpenMP, although it is yet at the development stage. It is built by just adding some code to the serial codes. The code is portable for GPU, CPU or their hybrid, hence can perform like OpenMP, MPI as well as GPU-based computation [18].

Besides these approaches, a few approaches that are intermediate options beyond CPU and GPU, i.e., Intel Xeon Phi bootable host processors; TPU [19]; and FPGA [20]. MPI and OpenACC approaches can easily adapt the implementation of Intel Xeon Phi. TPU is at an early stage of development and FPGA are not cost effective. Hence, these three approaches are not much popular.

2.3 Parallelization Models

The processors included in parallel computing cannot simultaneously perform interaction and exchange information. They communicate *via* specific strategies known as 'parallel models' based on network topologies as shown by Fig. 1. These parallel models are classified into four main communication strategies [21,22]: (i) Master-slave; (ii) Coarse-grained (island model); (iii) Fine-grained (cellular model) and; (iv) Hybrid.

The master-slave model works like the star topology. Fitness evaluations work in parallel on the slave processors, and the master processor controls the operations and generations of all n slave processors as shown in Fig. 1a. Coarse-grained and fine-grained models explore some neighborhood restrictions for communications' between processors. Fine-grained models are also known as cellular models, master-slave models are also known as star topology and coarse-grained models are also known as island models or ring topology. This makes the corresponding algorithm more robust and efficient. In the coarse-grained model, whole population is divided into n mutually independent sub-populations that represent a processor unit, called 'island' as shown in Fig. 1b. Some individuals are exchanged among islands according to some migration strategy (exchange of individuals is called migration). This communication model is controlled by several parameters like: migration strategy, migration population, and size of sub-populations. In a fine-grained model, individuals are arranged in a 2D grid in which each individual has 4 neighbors as shown in Fig. 1c. Communication between these neighborhoods may occur in several ways; hence, information exchange is delayed between non-neighbor processors. Hybrid models are the hybridization of two or more above-discussed models.

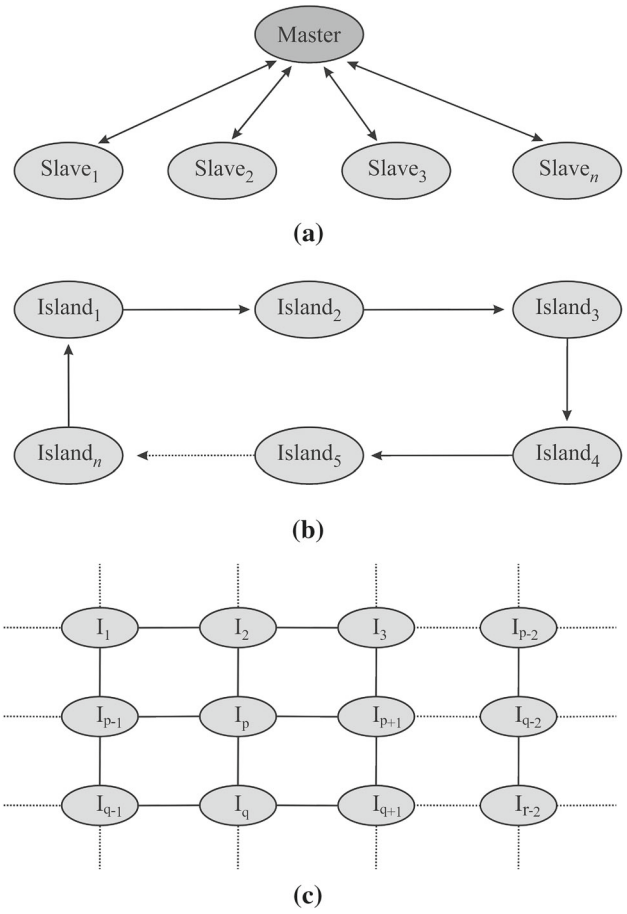

Fig. 1 Parallel models in PPSO **a** Master-slave, **b** coarse-grained, **c** fine-grained

2.4 Conventional Parallel PSO Algorithms

PSO is enriched with inherent parallelism. The particles in a swarm proceed in parallel, but the interactions of the particles remain non-simultaneous. This interaction determines the *gbest* and *pbest* positions for velocity and position update. During this procedure, the communication could be within the complete swarm or between the sub-groups of particles within swarm named as sub-swarms. The way of communication between the sub-swarms/nodes provides four basic types of PPSO variants, that include: (i) Star PPSO (ii) Migration PPSO (iii) Diffusion PPSO (iv) Broadcast PPSO. For multiprocessor parallelization, these variants are presented by Fig. 2a–d. Each processor represents a sub-swarm of PPSO.

2.4.1 Star PPSO

This variant of PPSO is also known as PPSO_star. Star PPSO is based upon master-slave topology. As can be observed by Fig. 2a, the communication occurs in a star shape, i.e., one sub-swarm named as 'master' (lying at the middle of the star) communicates the information between all the remaining sub-swarms named as 'slaves' (lying at the edges of the star). No direct communication between the slaves occurs in this process. The Star PPSO variant works as follows:

Step 1: The master determines and shares all the algorithm parameters to the slaves. These parameters include number of iterations, inertia weight, communication period, population size and the acceleration coefficients.
Step 2: Each sub-swarm evolves separately and obtains its *pbest* and *gbest*.
Step 3: Then, all the other sub-swarms called slaves communicate their *pbest* information to the master node. This process occurs at a certain communication period.
Step 4: The master determines the *gbest* and communicates this information to all the slaves.
Step 5: Each sub-swarm updates its velocity and position.
Step 6: Again each slave communicates the information about its *pbest* to the master and the master determines the new *gbest*.
Step 7: The process continues until the termination criteria is achieved.

2.4.2 Migration PPSO

Also known as circle_PPSO and ring_PPSO, this variant assumes that each sub-swarm can communicate with its neighboring sub-swarms in the circular area only, as presented by Fig. 2b. Hence, the information of a sub-swarm can be conveyed to the sub-swarms of the circle existing at the left and right positions. The process is similar to the process of coarse-grained parallel models in previous subsection. The process of migration PPSO is as follows:

Step 1: All algorithm parameters are predetermined.
Step 2: Each sub-swarm evolves separately and obtains its *pbest* and *gbest*.
Step 3: The best particle of each sub-swarm is migrated to the neighboring sub-swarm at a certain communication period to replace the worst particle of the sub-swarm. *gbest* is also updated with each communication.
Step 4: Sub-swarms update their positions and velocities with updated *pbest* and *gbest*.
Step 5: Repeat *Step* 3.
Step 6: The process continues until the termination criteria is achieved.

2.4.3 Broadcast PPSO

Also known as share_PPSO, it allows all the sub-swarms to communicate with all the other sub-swarms, as presented in Fig. 2c. As the name suggests, all the sub-swarms communicate and execute in parallel. Each information is broadcasted to all the sub-swarms. *Steps* 1 and 2 of this variant remain same as in migration PPSO. The remaining process works as follows:

Step 3: Then, all the sub-swarms share their *pbest* position information to obtain *gbest* of the swarm at a certain communication period.
Step 4: With updated *pbest* and *gbest*, sub-swarms update their positions and velocities.
Step 5: *Step* 3 is repeated.
Step 6: The process continues until the termination criteria is achieved.

2.4.4 Diffusion PPSO

Based on the fine-grained topology, diffusion PPSO is basically similar to the migration PPSO, except the number of communicating neighbors. As shown by Fig. 2d, the number of neighbors in the communication of each sub-swarm becomes four in place of two, that is the sub-swarms existing at left, right, up and down positions. Remaining sub-swarms at the corners of the rectangle are not included into the communication to the sub-swarm at the middle. Rest complete process remains same as that in migration PPSO.

3 Comprehensive Survey on Parallel PSO

This section presents the concise review of all the studies performed on PPSO so far. The summary of all these studies is provided in Table 1. The studies are classified on the basis

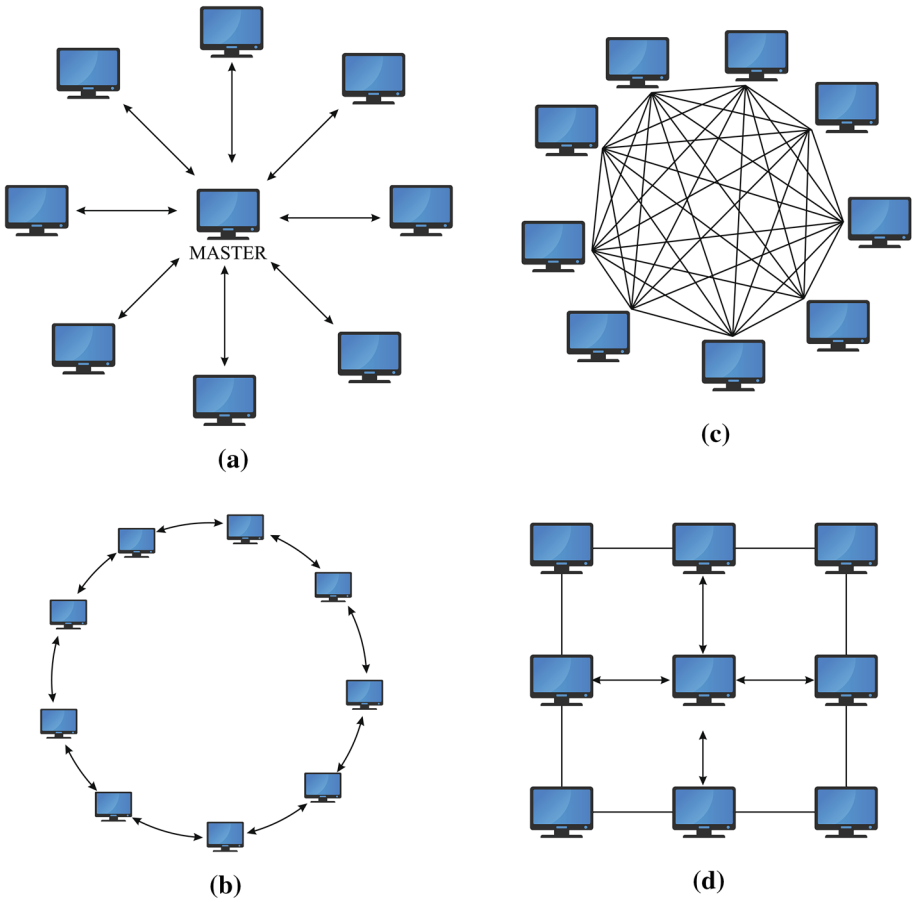

Fig. 2 Conventional PPSO variants **a** star PPSO, **b** migration PPSO, **c** broadcast PPSO, **d** diffusion PPSO

of: PSO variant developed/adopted; year of publication; type of parallelization along with parallelization model; the area for which the variant is applied and the objective of the study. All the abbreviated terms of the table related to PPSO version are provided in expanded form in the corresponding text, whereas the basic abbreviations are provided after abstract. The work is classified on the basis of CPU-based parallelization and GPU-based parallelization. The implementations of PPSO algorithms parallelized on CPU and GPU are further classified at five criteria that include:

- *Algorithmic approaches,* i.e., the implementation has basically developed PPSO variant.
- *Application-based approaches,* i.e., the parallel version of well-known sequential PSO algorithms is proposed, along with its implementation on complex applications.
- *Communication topologies and parameters setting based approaches,* i.e., the work is basically dedicated for testing over communication topologies and optimal settings of parameters.
- *Hybridized approaches,* i.e., the approaches are hybridized to obtain an enhanced parallel PSO algorithm.
- *For multi-objective problems,* i.e., the algorithm is specifically developed to solve multi-objective problem.

3.1 CPU-Based Parallelization

3.1.1 Algorithmic Approaches

The first implementation in this category is by Gies and Yahya [23] in 2003. They implemented the algorithm containing a swarm with 10 agents, hence on 10 independent nodes of a Beowulf cluster. The algorithm converged the solution 8 times faster than their serial implementation. Further, Schutte et al. [24] implemented master-slave communication model, in which the master node exclusively performs the algorithm operations. The slave nodes perform the particle fitness evaluation on the design configurations received *via* MPI. The information exchange between master and slave nodes regarding the fitness and particle positions is performed at the end of a swarm movement cycle. The same group of authors with Reinbolt, evaluated parallel PSO in the very next year (Schutte et al. [25]) for two kinds of optimization problems for acquiring multiple local minima: large-scale analytical test problems and; medium-scale biomechanical system identification problems. They aimed to evaluate the improvement in algorithms' performance and convergence, due to the parallelization and increase in population size. In the very next year, Cui and Weile [26] designed syn-

chronous PSO as a parallel version of asynchronous PSO. In this scheme, all the particles update their position synchronously and then communicate each other for the current global best position. Now, they again synchronously update their position and velocity based on their own best position and global best position.

On the contrary, few asynchronous versions of parallel PSO were also suggested that includes an asynchronous parallel PSO by Venter and Sobieszczanski [27]. They aimed to improve the performance of previously proposed synchronous PSO. Synchronous PSO may lead to poor parallel speedup whenever: number of particles is not an integer multiple of the number of processors; the parallel environment is heterogeneous or; the analysis time increases with respect to the design point being analyzed. In synchronous implementation, each particle in the parallel process waits for all the particles to complete the process before moving to the next iteration. But, in asynchronous implementation, no particle waits for others to complete the process; hence, no idle processors are left during the process. The parallel efficiency gets greatly improved. Similar kind of synchronous PSO was proposed by Chusanapiputt et al. [28], as a synchronous implementation accompanied by the relative velocity updating based parallel relative PSO (PRPSO). In this strategy, after exploring all the neighborhood, the slave sends its best position and corresponding velocity to the master. A subset of the best velocities (that is providing best solutions) is selected by the master, and the next move is decided accordingly. Further, in this trend, Koh et al. [29] added point-to-point communication strategy in parallel asynchronous PSO (PAPSO) algorithm implemented in homogeneous and heterogeneous computing environments. The results of asynchronous version were compared with synchronous version, in which results over robustness and convergence rate were comparable for both, whereas in parallel performance asynchronous version was significantly better than synchronous version. As an unique implementation, McNabb et al. [30] developed PSO as MapReduce (MRPSO) parallel programming model in Hadoop. In the mapping phase, particle is mapped and obtains updated velocity, position and $pbest$. Further, in the reduce phase, all the information is collected for the swarm and $gbest$ gets updated.

Alike other synchronous PSO versions, PPSO from Liu et al. [31] provided parallelization between particles, naming position and velocity updation as a sub-process. The optimal particle of each sub-process, i.e., slave, moves to the main process, i.e., master. Then the main process determines optimal particle and broadcasts the information to each sub-process. Hence, proposed two-phase PPSO (TP_PPSO) divides the search into two-phase optimization. In the first phase, individual orientation factor function uses exploration search area. In the second phase, overall orientation factor function uses expanded search area. Further, Han et al. [32] included constraint handling in the proposed version for motion parameter estimation. They included the constraints into the objective functions and further solved them for the combination that provides best possible solution. An asynchronous version entitled parallel multi-population PSO (PMPSO) is proposed by Wang et al. [33]. PMPSO gets randomly initialized alike other versions. The particles get ranked as per their performance evaluated at the fitness function, and then, the sub-populations are created. The best position in population, as well as in the sub-population, is considered for position and velocity updating. Jeong et al. [34] proposed a PPSO model for PC-cluster that exchanges the information between the sub-populations one by one, i.e., with coarse grain topology. In order to maintain the swarm diversity and to avoid premature convergence, Lihua et al. [35] applied client-server system-based parallel computation software system for distributed cascade optimization dispatching. The communication remains asynchronous and algorithms takes migration strategy into consideration so as to choose appropriate individual for exchange and migration. This implementation reports more accurate results, speed up in calculations and improvement in the convergence performance. The prospect of utilizing multiple swarms in n-dimensional design spaces concurrently in parallel computing environments was investigated by Kalivarapu et al. [36], through developing PSO variant with digital pheromones. With the increase in problem dimension, the speedup and parallel efficiency of the algorithm gets significantly improved. Singhal et al. [37] simply implemented asynchronous PSO via MPI commands/functions on the multiple processes. The algorithm derives the process of splitting particles in the finest way for every number of processors, and then, the processor with best results becomes the root processor at the end of each cycle.

An approach similar to the master-slave strategy containing two kinds of agents was proposed by Lorion et al. [38]. They adapted agent-based structure for distributing and managing a particle swarm on multiple interconnected processors. Agent-based parallel PSO (APPSO) includes one coordination agent that coordinates between swarms and other multiple swarm agents. Further, Farmahini-Farahani [39] presented a hardware pipelined PSO (PPSO core) for performing computational operations of the algorithm for numerous types of discrete optimization problems with the notion of system-on-a-programmable-chip. The parallel asynchronous PSO implementation followed the process alike in [27]. They employed a non-chip multiprocessing architecture for evaluating the fitness in parallel by utilizing multiple embedded processors. For hiding the communication latency, Li and Wada [40] proposed globally synchronized parallel PSO (GSP PSO) with delayed exchange parallelization (DEP). The algorithm extracts inherent parallelism of PSO, by overlapping communication with computation.

DEP helped in alleviating temporal dependencies, existing in the iterations of the algorithm; hence, GSP PSO became tolerant to network delays. Basically, the algorithm delays the partial best fitness exchange to one loop later.

As an Hadoop implementation, Aljarah and Ludwig [41] proposed MapReduce-based parallel PSO clustering algorithm (MRCPSO) for data-intensive applications, tested on different sized large-scale synthetic data sets. The algorithm aimed at optimal clustering in three sub-modules. First sub-module updates, the particle swarm centroids in MapReduce. In second sub-module, the fitness gets evaluated for the new particle centroids. In the third module, the merging occurs for all the updated fitness values along with the updating in the personal best and global best centroids. Parsopoulos [42] presented parallel cooperative micro-PSO (PCOMPSO), established upon the disintegration of the original search space in subspaces of smaller dimension. Two types of computer systems were engaged, i.e., academic cluster and desktop multi-core system for evaluating the approach. The solution is claimed to achieve quality in results as well as superior runtime. Gulcu and Kodaz [43] proposed parallel comprehensive learning PSO (PCLPSO) algorithm which has the characteristics of having multiple swarms that work cooperatively and concurrently. The local best particles of the swarm gets exchanged in every migration process so as to maintain the diversity of the solutions. For obtaining higher solution quality from PPSO, Zhang et al. [44] implemented the local model PSO (LPSO), global model PSO (GPSO), comprehensive learning PSO (CLPSO) and the bare bone PSO (BPSO) on different slave processors. Further, an in-depth investigation and evaluation of the parallel design and pursuit of a parallel PSO-back-propagation (BP) neural network algorithm was conducted by Cao et al. [45]. The work optimizes the initial weights and thresholds for the BP neural network. The performance is assessed on image repository from the SUN database scene image library. Tian et al. [46] presented a parallel co-evolution structure of quantum-behaved PSO (PC_QPSO) with a revised differential grouping approach to break up the high-dimensional problems into sub-problems. The sub-problems get optimized individually with intermittent communication that enhances the resulting quality without hiatus in the connection between interacted variables.

For evaluation at many and multi-core architecture, Nedjah et al. [47] presented fine-grained parallel PSO (FGP-PSO), implemented over both the architectures (many-core and multi-core), along with testing on a serial implementation. The termination criteria was taken as leaning upon the acceptability of the solution. The effect of fine-grained parallelism on the convergence time of high-dimensional optimization was also studied. Atashpendar et al. [48] proposed cooperative co-evolutionary speed-constrained multi-objective PSO (CCSMPSO) along with scalability analysis. The scalability analysis performed on Intel Xeon L5640 contained two studies: tendency of scale of the algorithms with varying problem size; scalability as a function of parallelization. Lai and Zhou [49] proposed parallel PSO based on osmosis (PPBO), implemented on numerical optimization problems. The algorithm is capable to obtain three parameters, i.e., migration interval, migration direction, and migration rate which are helpful to determine when, from which sub-population to which sub-population, and how many particles will be migrated.

3.1.2 Application-Based Approaches

In application-based approaches, PPSO has been implemented on miscellaneous problem areas. Ying et al. [50] addressed DORPD problem by proposing PPSO algorithm that divides the problem into sub-problems as concurrent processes. The algorithm is evaluated on test cases of IEEE power systems, containing reactive power sources, time-varying loads with tap position of transformers, and control over generator terminal voltages. Further, Subbaraj et al. [51] solved extensive ED problems by modified stochastic acceleration factors (PSO-MSAF). PSO-MSAF is based upon macro evolution, i.e., creation by multiple populations, whereas, conventional PSO is based upon micro evolution, i.e., creation by single population. Further, the six learning parameters of PSO (i.e., c_1, c_2, upper and lower limit for random cognitive and social learning parameters) are uniquely determined for each swarm. Further, Li and Chen [52] proposed a parallel PSO color quantization algorithm to determine the most superior palette and to quantize color mapping of every pixel. They implemented parallelization on all the 'for' loops.

Moving in the same trend, Prasain et al. [53] conceptualized PSO for the option pricing problem and composed a sequential PSO algorithm followed by parallel PSO for shared memory machine employing OpenMP, distributed memory machine adopting MPI and homogeneous multi-core architecture running hybrid of MPI/OpenMP. Similarly, Qi et al. [54] adopted data parallelization in PSO to solve one-dimensional inverse heat conduction problem by using implicit difference method. Drias [55] designed two novel PSO algorithms (sequential and parallel) for web information retrieval (IR) with direct search method. They implemented PSO modeling that considers the documents identifiers for obtaining the particles positions, except for the evaluation process, which requires the content of document identifiers. The indexing in the documents is performed by the vector space model. Torres and Castro [56] implemented Local PSO (LPSO) in the parallel environment with DC network model for Garver and IEEE 24-bus networks. This local version of PSO is aimed to take advantage of exploration capability of the algorithm. In this approach, each particle communicates

its best position only to the neighboring particles, not to the swarm. Omkar et al. [57] proposed a parallel version of the VEPSO based on the peer-to-peer architecture for the optimized design of laminated composite plates, a combinatorial explosive constrained nonlinear optimization problem. The parallel approach shows accelerate for adequate numbers of processors and scalable for an enlarged number of particle size and populations.

For achieving better prediction of the stock price trend, Wang et al. [58] presented time-variant PSO (TVPSO) algorithm and considered the complex performance-based reward strategy (PRS) for trading. Satapathy et al. [59] stimulated convergence rate by taking communication of particles, fault tolerance, and load balance into consideration. The server is treated as the nucleus of data interchange for dealing with agents and managing the partaking of global best position among the distinctive clients. Further, Xu et al. [60] proposed parallel adaptive PSO for optimizing the parameters and selecting the features of support vector machine (PTVPSO-SVM). The approach designs the objective function with weights on the average accuracy rates, number of support vectors and the selected features. The implementation contains features like: adaptive control parameters with time changing acceleration coefficients; inertia weight to regulate the local and global search; mutation operators to overcome the problem of premature convergence. In a similar way, Mohana [61] proposed position balanced parallel PSO (PB-PPSO) for resources allocation with profit maximization and increased user satisfaction level in the cloud computing environment. The algorithm is claimed to overcome the issues of machine learning methods SVM and ANN for the addressed problem. Chen et al. [62] proposed a method, to simultaneously include the number of support vectors, the number of features in objective function and the average classification accuracy rates to achieve the maximum generalization capability of SVM. Gou et al. [63] proposed multi-swarm parallel multi-mutation PSO (MsP-MmPSO) for the parallel derivation of association rules. Also, they found a reduction in computation time of the algorithm by implementing a good task allocation method in multi-CPU parallel computation environment. Govindarajan et al. [64] designed a PPSO clustering algorithm for learning analytics platform, along with experiments on real-time data. The learner's data manifested as big data have accuracy, efficiency, and an ability for understanding the learner's competence. Fukuyama [65] evaluates fast computation by parallelization and dependability of parallel PSO for Volt/Var Control. Volt/Var Control is needed to trim the control interval and work with larger-scale power systems.

For solving conditional nonlinear optimal perturbation (CNOP), Yuan et al. [66] proposed sensitive area selection-based PSO (SASPSO). CNOP are the problems with high-dimensional complex numerical models, useful in climate prediction and studying the predictability of numerical weather. SASPSO was implemented on Zebiak-Cane (ZC) numerical model. Kumar et al. [67] used heterogeneous multiprocessor systems for: minimizing schedule length with constraint of energy consumption and minimizing energy consumption with the constraint of schedule length. PPSO was implemented for obtaining an energy efficient schedule and optimal power supply. Further, Moraes et al. [68] proposed an asynchronous and immediate update parallel PSO (AIU-PPSO) by revisiting the asynchronous parallelization of PSO with pseudo-flight and weighted mean position based stop criterion. It was successful in solving an actual parameter estimation problem of a population balance model, enriched with a high-cost objective function and 81 parameters for estimating. Kusetogullari et al. [69] proposed parallel binary PSO (PBPSO) algorithm for developing a robust to illumination changes, unsupervised satellite change-detection method. Multiple BPSO algorithms were run simultaneously on different processors for finding the final change detection mask. Parallel mutation PSO (MPSO) was proposed by Jia and Chi [70] for optimizing the soil parameters of the Malutang II concrete face rockfill dam. A parallel finite element method was implemented for the fitness evaluation of the particle swarm. The objective remained to minimize the deviation between the prototype monitoring values and to compute earth-rockfill dam displacements. Fukuyama [71] investigated the dependability of PPSO-based voltage reactive power control on IEEE 14, 30, and 57 bus systems. The method was found to maintain the quality of solutions, despite large fault probability when a pertinent number of maximum iteration is used for temporary faults. To distribute the workload of a parameter estimation algorithm to parallel connected computing devices, Ma et al. [72] proposed PSO-based parameter estimation algorithm. Parallel PSO was found outperforming to the sequential PSO. Hossain et al. [73] proposed parallel clustered PSO (PCPSO) and k-means-based PSO (kPSO) to optimize service composition process. Five QoS attributes were considered in objective function, i.e., reliability, availability, reputation, computation time and computation cost.

To elaborate and model the batch culture of glycerol to 1,3-propanediol by Klebsiella pneumoniae, Yuan et al. [74] presented nonlinear switched system which is enzyme-catalytic, time-delayed and contains switching times, unknown state-delays, and system parameters. The process contains state inequality constraints and parameter constraints accompanied by calibrated biological robustness as a cost function. To extract and estimate the parameters of the PV cell model, Ting et al. [75] proposed parallel swarm algorithm (PSA) that minimizes root-mean-square error that seeks the difference between determined and computed current value. They coded the fitness functions in OpenCL kernel and executed it on multi-core CPU and GPUs. Further, Liao et al.

[76] proposed multi-core PPSO (MPPSO) for improving the computational efficiency of system operations of long-term optimal hydropower for solving the issue of speedily growing size and complexity of hydropower systems. The algorithm claims to achieve high-quality schedules for pertinent operations of the hydropower system. Li et al. [77] used parallel multi-population PSO with a constriction factor (PPSO) algorithm to optimize the design for the excavator working device. They authenticated kinematic and dynamic analysis models for the hydraulic excavator. Luu et al. [78] proposed a competitive PSO (CPSO) to improve algorithm performance $w.r.t.$ stagnation problem and diversity. They used travel time tomography algorithm and tested it on real 3D data set in the context of induced seismicity on four Intel Xeon Platinum 8164 CPU. Nouiri et al. [79] implemented two multi-agent PSO models, i.e., MAPSO1 and MAPSO2 to solve FJSP. The benchmark data for testing were taken from partial FJSP and total FJSP. Yoshida and Fukuyama [80] presented parallel multi-population differential evolutionary PSO (DEEPSO) for voltage and reactive power control (VQC), tested on IEEE bus systems. The problem formulation was done as a mixed integer nonlinear optimization problem with upper & lower limits on bus voltage and upper limit on line power flow.

3.1.3 Communication Topologies and Parameters Setting Based Approaches

Inspired by parallelism for the data, Chu and Pan [81] presented three communication strategies in PPSO that are based on the strength of the correlation of parameters, i.e., for loosely and strongly correlated parameters and for unknown parameters. First strategy migrates the best particle to each group and mutates it for replacing the poorer particles in specific number of iterations, whereas second strategy migrates the best particle to its neighboring group for replacing the poorer particles in specific number of iterations and third strategy divides the group into two subgroups. Then applies first communication strategy to first subgroup and second communication strategy to second subgroup in specific number of iterations. Waintraub et al. [82] simulated multiple processors and evaluated communication strategies in multiprocessor architectures for two intricate and time-consuming nuclear engineering problems. Further, they proposed neighborhood-Island PPSO (N-Isl-PPSO) (based on ring and grid topology), and the island models (based on ring and island topology) as the communication strategy tested over several benchmark functions. The outcome of communication strategies based PPSO was evaluated in terms of speedup and optimization outcomes. Sivanandam and Visalakshi [83] proposed parallel orthogonal PSO (POPSO) along with versions like: PSO with fixed & inconstant inertia, MPSO, PPSO, elitism enabled PSO, hybrid PSO, orthogonal PSO (OPSO) and parallel OPSO (POPSO) for scheduling heterogeneous processors for heterogeneous tasks using dynamic task scheduling. They implemented data parallelism as well as task parallelism. In data parallelism, the data splitted into smaller chunks are operated in parallel, whereas, in task parallelism, different tasks are run in parallel. As an extension on the communication topologies, Tu and Liang [84] developed a PSO model whose particles concurrently communicate with each other. The particles in a swarm are separated into several subgroups that communicate other subgroups by parallel computation models based on network topologies, i.e., broadcast, star, migration, and diffusion. The results of sequential and parallelization enabled different network topologies got compared. To simplify and save the cost of parallelization, multiple threads were used for concurrent communication of particles.

3.1.4 Hybridized Approaches

Zhang et al. [85] developed hybrid moving boundary PSO (hmPSO), a hybrid of effectiveness of NelderMead (NM) methods and basic PSO for local and global searching respectively. NM methods directly evaluate the objective function at multiple points within the search space. The parallel implementation was performed on linux cluster 'Hamilton' with 96 dual-processor dual-core Opteron. Roberge et al. [86] hybridized GA and PSO to manage the complexity of the UAVs with dynamic properties and evaluate the trajectories in a complex 3D environment that are feasible and quasi-optimal for fixed wing. The cost function is formed by penalization of longer paths, and also by the penalization of the paths, with greater average altitude, undergo danger zones, bumping with the ground, necessitating for additional fuel than the available in UAV at beginning, demanding further power than the maximum power at hand and the paths that cannot be smoothed using circular arcs. Jin and Rahmat-Samii [87] combined PSO and the finite-difference time-domain (PSO/FDTD) with master-slave strategy. The master node tracks the particles, updates their position and velocities, and collects the simulation results, whereas each slave node contain a particle that performs FDTD fitness evaluation. Han et al. [88] implemented PPSO for PID controller tuning in the application position of high real-time necessity and control accuracy.

A hybrid of variable neighborhood search and PSO was proposed by Chen et al. [89] as VNPSO. The formulated problem of multi-stage hybrid flow shop scheduling is a mixed integer linear programming problem. The work addresses both sequence-independent as well as sequence-dependent setup time. Soares et al. [90] proposed scheduling of V2G in smart grids considering an aggregator with different resources, emphasizing on distributed generation and V2G, and also adding the opinion of the electric vehi-

cles (EVs) possessor. They considered a case study of the 33-bus distribution network that contains 32 loads, 1800 EVs, 66 distributed generation plants and 10 energy suppliers. Yuan [91] performed advancement in PSO with tabu search algorithm and then parallelized cooperative co-evolution based PSO (PCCPSO) in Zebiak-Cane model for solving CNOP problem. Cao et al. [92] proposed parallel cooperative co-evolution PSO (PCCPSO) for solving high-dimensional problems in parallel. They combined the probability distribution functions, i.e., Gaussian distribution, Cauchy distribution and Levy distribution. They also combined global and local versions of PSO for space exploration and for speeding up the convergence. Hence obtained hybrid algorithm was implemented in Spark platform. Long et al. [93] employed local search and global search neighborhood search strategies into quantum-behaved PSO and introduced parallel quantum-behaved PSO with neighborhood search (PNSQPSO), to increase the diversity of the population; and parallel technique for reducing the runtime of the algorithm. Peng et al. [94] proposed three multi-core PPSO algorithms, i.e., PPSO_star, PPSO_ring, and PPSO_share, based on Fork/Join framework with concurrency in Java. Proposed algorithm can interchange information between the threads (sub-swarms). Fork/Join framework assigns threads to different CPU cores, whereas synchronization-and-communication mechanisms are employed for exchanging information among the threads.

3.1.5 For Multi-objective Problems

Vlachogiannis and Lee [95] implemented synchronous PSO enriched with vector evaluated version, i.e., VEPSO for multi-objective optimization problems. This variant contains the number of objective functions equal to the number of swarms, working in parallel. Fan and Chang [96] added a remarkable progress in the PPSO implementations, by proposing parallel particle swarm multi-objective evolutionary algorithm (PPS-MOEA). PPS-MOEA was established upon the idea of Pareto dominance and state-of-the-art parallel computing for improving algorithmic effectiveness and efficiency. In this multi-swarm algorithm, after basic PSO operations, each swarm shares a fixed number of less crowded members after specific migration period. External archive keeps on getting updated after each cycle for a fixed number of non-dominated solutions. Vlachogiannis and Lee [97] implemented parallel vector evaluated PSO (parallel VEPSO) and applied it to compute reactive power control by formulating a MOP. The number of swarms is taken equal to the number of objectives and each swarm works to optimize corresponding single objective. Li et al. [98] proposed decomposition based multi-objective PSO (MOPSO/D) that uses both MPI and OpenMP to implement the algorithm with a hybrid of distributed and shared memory programming models. Borges et al. [99] implemented PSO to a large-scale nonlinear multi-objective combinatorial resources scheduling problem of distributed energy, including a case study of a 201-bus distribution real Spanish electric network from Zaragoza. Single objective function was formed by the weighted sum of two objectives, i.e., to maximize the profit and minimize CO_2 emission.

3.2 GPU-Based Parallelization

3.2.1 Algorithmic Approaches

A fine-grained parallel PSO (FGPSO) was proposed by Li et al. [100] in 2007 for maintaining population diversity, inhibiting premature solutions and keeping the utmost parallelism. The algorithm basically maps FGPSO algorithm to texture-rendering on consumer-level graphics cards. Further approach is found from 2009 that includes implementation of standard PSO on GPU, i.e., GPU-SPSO by Zhou and Tan [101]. They executed SPSO on both GPU and CPU. The running time was found greatly shortened and running speed became 11 times faster in comparison with SPSO on CPU (CPU-SPSO). Further, Hung and Wang [102] proposed GPU-accelerated PSO (GPSO) by implementing a thread pool model with GPSO on a GPU, aimed to accelerate PSO search operation for higher dimension problems with large number of particles. The authors focused on addressing the box-constrained, load-balanced optimization problems by parallelization on GPU.

Further, Zhu et al. [103] proposed a parallel version of Euclidean PSO (pEPSO) for better convergence and accelerating the process, since EPSO requires long processing time in massive calculations. The algorithm employed fine-grained data parallelism to calculate fitness with GPU. Kumar et al. [104] presented a study that finds a remarkable reduction in execution time of C-CUDA implementation over sequential C. The algorithm divides the population into one-dimensional sub-populations and then individual vector tries to optimize corresponding sub-population. Calazan et al. [105] proposed parallel dimension PSO (PDPSO) for GPU implementation, where each particle gets implemented as a block of threads and each dimension get mapped onto a distinct thread. The allotment of the computational tasks becomes at a finer degree of granularity. Shenghui [106] proposed an algorithm that expedites the rate of convergence of the particle swarm by employing a large amount of GPU threads to deal with each particle. The algorithm applies CA logic to the PSO; hence, a particle is considered as a CA model. The number of threads in GPU remains equal to the number of particles. Independent calculation space is provided for each particle on respective thread. Further, Li et al. [107] implemented PPSO algorithm on CUDA in AWS that can be employed on any network connected computer.

They run all the processes in parallel: evaluation of the fitness values and update process (of the current position, velocity, particle best fitness and global best fitness).

Use of coalescing memory access, for a standard PSO (SPSO) on a GPU, based on the CUDA architecture was performed by Hussain et al. [108]. The implementation on GPU was found 46 times faster than CPU serial implementation. In coalescing memory access, video RAM memory is used, which is quite efficient in simultaneous memory access by threads in a warp. Wachowiak et al. [109] adapted PSO for difficult, high-dimensional problems and proposed adaptive PSO (APSO) for parallelization on already accessible heterogeneous parallel computational hardware that contain multi-core technologies speeded up by GPU and Intel Xeon Phi co-processors expedited with vectorization. Task-parallel elements are carried out with multi-core parallelism, while data-parallel components get executed *via* co-processing by GPUs or vectorization.

3.2.2 Application-Based Approaches

A PPSO-based parallel band selection approach for HSI was proposed by Chang et al. [110]. GPU-CUDA, MPI and OpenMP were used to take advantage of the parallelism of PPSO and to constitute a set of near-optimal greedy modular Eigenspaces (GME) modules on all parallel nodes. The outperformance of PPSO was judged on land cover classification simulator MODIS/ASTER airborne (MASTER) HSI. Mussi et al. [111] proposed PPSO-based road sign detection approach that detects road sign shape and color. The algorithm simultaneously detects whether a sign belongs to a certain category and estimates its ideal location in accordance with the camera reference frame. Each GPU thread contains corresponding position and velocity of a particle. Similarly, Liera et al. [112] designed PPSO for low-cost architecture, i.e., general-purpose GPU (GPGPU). Each thread runs its own function evaluation simultaneous to the other threads. Similarly, other implementations are found in this category. Roberge and Tarbouchi [113] proposed parallel CUDA-PSO for minimization of harmonics in multilevel inverters by utilizing the computational potential of GPU attached with the parallelism for real-time calculation of most favorable switching angles. Rabinovich et al. [114] introduced a tool that is able to solve the complex optimization problems with discontinuities, nonlinearity, or high dimensionality. Proposed powerful tool parallel gaming PSO (GPSO) can be implemented effectively on a GPU. Datta et al. [115] implemented CUDA version of PSO (CUDA PSO) for inverting self-potential, magnetic and resistivity data of a geophysical problem. The results of CUDA PSO were compared to CPU PSO, and a significant speedup was obtained from CUDA PSO compared to a CPU only version, maintaining the same quality of results. Dali and Bouamama [116] proposed a solution to the maximal constraint satisfaction problems (Max-CSPs) by introducing parallel versions for GPU, i.e., parallel GPU-PSO for Max-CSPs (GPU-PSO) as well as GPU distributed PSO for Max-CSPs (GPU-DPSO). CSP solution should be a complete set of values that satisfy all the constraints, which is considered an NP-hard problem. Further, serial PSO for graph drawing (SPGD) and parallel PSO for graph drawing at vertex level (V-PGD) was proposed by Qu et al. [117]. A force-directed method was used for positioning the vertices, in which each particle corresponds to a layout of the graph. Since the energy contribution is the sum of attractive and repulsive forces, so it will be low if adjacent vertices in the original graph are close to each other. Lorenzo et al. [118] proposed PPSO algorithm for hyper-parameter optimization in DNNs. The particle population represents a combination of hyper-parameter values. Training and testing experiments are performed on MNIST data sets of handwritten digits. Liao et al. [119] proposed distributive PSO (DPSO) to address luminance control problem, formulated as a constrained search problem. DPSO partitions the population of particles into groups with employment in GPU and Hadoop MapReduce. Zou et al. [20] proposed OpenMP and CUDA based PPSO and parallel GA (PGA). The algorithms were implemented on FPGA. Further, the implementation of FPGA-based parallel SA was done for JSSP.

3.2.3 Communication Topologies and Parameters Setting Based Approaches

A comparative study of parallel variants for PSO on a multithreading GPU was performed by Laguna-Sanchez et al. [120]. Three communication strategies with four PSO variants were tested so as to obtain significant improvement in performance of PSO. The communication strategies include: master-slave, island and diffusion. The PSO variants include: sequential, $\text{Global}_{\text{ev}}$, $\text{Global}_{\text{ev}}$+up and embedded variant. In $\text{Global}_{\text{ev}}$ variant, the objective function evaluations get parallelized, whereas, in $\text{Global}_{\text{ev}}$+up variant, all processes get parallelized that include updation of: fitness function evaluation, velocity, position, and inertia. The embedded variant hybridizes both the variants and runs complete PSO on the GPU, assuming it as a black box. Mussi et al. [121] discussed possible approaches to parallelize PSO on GPU, i.e., multi-kernel variant of PPSO with the global, random and ring topology. In order to eliminate the dependence between particles' updates, synchronous PSO is implemented with updation of global and local best at the end of generation. RingPSO tries to relax the constraint of synchronization and allows the load of computation get allocated over all streaming multiprocessors. Altinoz et al. [122] presented a comparison in the execution time of PPSO for successively increasing population sizes and problem dimensions. They compared the execution time of the proposed chaotic dis-

tributed population-based version chaotic P-PSO (CP-PSO) with uniformly distributed population-based version. Nedjah et al. [123] discussed the impact of parallelization strategy and characteristics of the exploited processors on the performance of the algorithm. They proposed cooperative PPSO (CPPSO) and mapped the optimization problem onto distinct parallel high-performance multiprocessors, based on many-core and multi-core formation employed in MPICH, OpenM, OpenMP with MPI and CUDA. Many-core GPGPU-based parallel architecture was found most efficient among the compared strategies. Wu [124] studied the effect of dimension, number of particles, size and interactions of the thread-block in the GPU versus CPU, on the computational time and accuracy.

3.2.4 Hybridized Approaches

Franz and Thulasiraman [125] parallelized the hybrid algorithm of multi-swarm PSO (MPSO) and GA on a hybrid multi-core computer with accelerated processing unit to improve performance, by taking advantage of APU-provided close coupling between CPU and GPU devices. Ge et al. [126] presented a joint method to inverse the relaxation time of longitudinal (T1)-transversal (T2) spectrum in a low field NMR, so as to obtain an optimal truncated position. The method is a combination of iterative TSVD and PPSO. Jin and Lu [127] proposed PPSO with genetic migration (PPSO_GM) so as to introduce a better converging algorithm for ten 100-dimensional benchmark test functions. They implemented selection, crossover and mutation operators on the particles in sequence. After completion of migration among swarms, new swarms run PSO independently.

3.2.5 For Multi-objective Problems

Zhou and Tan [128] parallelized VEPSO for GPU-based MOP. The GPU-based parallel MOPSO versions have been compared with CPU-based serial MOPSO. The work is an extension of author's previous work [101]. For proposed algorithm they considered only the non-dominated solutions of the final population, instead of the entire evolution process that helps in reducing the data transfer time between GPU and CPU, hence it speeds up the process. Similarly, Arun et al. [129] implemented MOPSO on GPU using CUDA and OpenCL. The performance is claimed to be improved by 90 % in comparison with sequential implementation. Use of archiving technique further improves the speedup.

4 Results and Discussions

Proposed work is summarized in Table 1 and Figs. 3, 4, 5. Figure 3 presents the publication analysis of PPSO based on the parallelization strategy. As can be observed from the figure, MPI is the most popular parallelization strategy with 34.64% share, whereas GPU has 28.18% share. The follower strategies in descending order include multi-core (11.82%), OpenMP (9.09%), Hadoop (6.36%) and MATLAB parallel computing toolbox with 6.36%. The other parallelization strategies with 4.55% share include PVM, CloudSim with virtual machine, OpenCL and Multi-threading. Moreover, 7% approaches implemented multi-objective optimization along with parallelization. Besides this, one study (Wachowiak et al. [109]) has implemented Intel Xeon Phi^{TM} co-processors and 22 studies have employed Intel Xeon processors with or without hybridizing with additional parallelization strategies. Figure 4 depicts the communication model-based distribution of the literature. The approaches with GPU or multi-core implementation which did not implement any communication strategies (due to the availability of default strategy) were excluded from this distributive analysis. As it can be observed that Master-slave approach is the most popular parallelization approach with 53.23% share, whereas coarse-grained approach has 27.42% and fine-grained has 14.52% share. The hybrid approaches with 4.84% share, present the hybridization of two or more communication models. Figure 5 presents the details of increasing popularity of PPSO in last decade. Although, nVIDIA^{TM} introduced CUDA^{TM} in 2006 [16] and MPI was introduced in 1992 [15] and PSO in 1995 [2], but PPSO got introduced to widespread research community 2009 onwards. As can be seen by the figure, the % of the publications on PSO before 2009 was merely 14% which got increased to 86% (from 2009 onwards). This demonstrates the growing reputation of PPSO.

5 Concluding Remarks and Future Work

The accelerated emergence of large-size real-world complex problems has raised the demand of parallel computing techniques. This has encouraged the research on heuristic optimization algorithms like PSO, a derivative-free prevalent swarm intelligence-based algorithm, particularly suitable to continuous variable problems, which is widely being applied to real-world problems. Implementation of parallel PSO with numerous parallelization models and strategies for solving complex applications has obtained significant attention by researchers.

In this conjunction, this paper presents a chronological literature survey on parallel PSO algorithm, its developed variants, implementation platform and strategy, applied area and the objective of addressed problem. Then, all the surveyed articles are further evaluated for concluding the popular parallelization strategy and communication topology. Despite of being user-friendly, GPU-based parallelization is very expen-

Table 1 Publications analysis on Parallel PSO

S. No.	PSO Variant	Year	Type of Parallelization + Strategy	Application	Objective Function
1	PPSO [24]	2003	MPI, Master-slave, Coarse-grained	To solve biomechanical system identification problem	Minimization of marker distance error
2	PPSO [23]	2003	MPI	To perform dual-beam array optimization	Fitness optimization of dual-beam array
3	PPSO [25]	2004	MPI, Coarse-Grained	To solve computationally demanding optimization problems	Two benchmark functions
4	Parallel synchronous PSO [26]	2005	MPI, Master-slave	Electromagnetic absorbers design	Griewank function
5	Parallel VEPSO [95]	2005	MPI, Ring topology	To determine generator contributions to transmission system	Optimization of the contributions to the real power flows
6	Synchronous and asynchronous PPSO [27]	2005	MPI, Master-slave	Design of an transport aircraft wing	Maximization of the aircraft flight range
7	PSO/FDTD [87]	2005	MPI, Master-slave	Multiband and wide-band patch antenna designs	To obtain best return loss and best bandwidth
8	PRPSO [28]	2005	MPI, Master-slave	To update Lagrange multipliers in a conventional Lagrangian relaxation method for constrained unit commitment problem	Maximization of the Lagrange function
9	Intelligent PPSO [81]	2006	Three correlation based communication strategies	To discuss performance of PPSO with three communication strategies	Three benchmark functions
10	PAPSO [29]	2006	MPI, Master-slave	Efficiently using the computational resources in the presence of load imbalance	Four small to medium-scale analytical test problems along with a medium-scale biomechanical test problem
11	FGPSO [100]	2007	GPU, Fine-grained	Solving constrained and unconstrained optimization problem	Texture-rendering on consumer-level graphics cards
12	MRPSO [30]	2007	Hadoop MapReduce, Fine-grained	Large-scale parallel programming	Radial basis function network
13	TP_PPSO [31]	2007	Multi-core, Island model	Economic data analysis and mining	Five benchmark functions
14	PPSO [32]	2008	MATLAB parallel computing tool box	To estimate motion parameters	Minimization of continuous constrained nonlinear optimization problem formulated with five-dimensions
15	PMPSO [33]	2008	OpenMP	Uncapacitated facility location	Transportation cost minimization
16	PPS-MOEA [96]	2009	MPI, Master-slave	Multi-objective problems	Eight benchmark functions
17	APPSO [38]	2009	Master-slave	Accelerates the optimization on an interconnected heterogeneous system	Three benchmark functions

Table 1 continued

S. No.	PSO Variant	Year	Type of Parallelization + Strategy	Application	Objective Function
18	PPSO [34]	2009	MPI	State estimation of power system	Minimization of the sum of the differences between the estimated and true values
19	PPSO [110]	2009	CUDA, MPI and OpenMP	Band Selection for HSI	GME cost function
20	Sequential, Global_ev, Global_ev+up and Embedded variants [120]	2009	GPU-based, Master-slave, island and diffusion approaches	To obtain the computing power of cluster in a conventional PC	Three benchmark functions
21	PPSO [50]	2009	MPI, Master-slave	For solving DORPD in power systems	Minimization of sum of network active power loss with the costs of adjusting the control devices
22	POPSO [83]	2009	Multi-core, Master-slave	Dynamic task scheduling	Minimization of the makespan of the schedule
23	GPU-SPSO [101]	2009	GPU	In high-dimensional problems and for special speed advantages for large systems	Four benchmark functions
24	PPSO [111]	2009	GPU, Coarse-grained	Road sign detection	A fitness function to detect a sign belonging to a certain category and to estimate its actual position
25	Parallel VEPSO [97]	2009	MPI, Ring topology	To obtain steady-state performance of power systems	Minimization of the real power losses and voltage magnitudes
26	N-Isl-PPSO [82]	2009	Master-slave, island and cellular topology, MPI	Reactor core design and optimization of fuel reload	Maximization of average thermal flux and cycle length
27	Coarse grain PPSO [35]	2009	MPI, Coarse-grained	To solve the multi-stage optimum mathematics model	Maximization of cascade generate power
28	Asynchronous PSO [37]	2009	MPI, Fine-grained	Parallelization of the asynchronous version of PSO	Benchmark functions
29	hmPSO [85]	2009	MPI	To enable calibration of geotechnical models from laboratory or field measurements	Five benchmark functions
30	PSO with digital pheromones [36]	2009	MPI, Coarse-grained	To improve the speedup and parallel efficiency	Six benchmark functions with varying dimensionality
31	PPSO [53]	2010	OpenMP, MPI and hybrid of MPI/OpenMP	Obtain optimal solution for American option pricing problem	Profit maximization
32	GPSO [102]	2010	Many-core, low-cost GPU	Tackling high-dimensional and complex optimization problems	Twelve benchmark functions
33	PPSO [52]	2010	OpenMP	Color quantization of image	Minimization of the Euclidean distance between pixels and particles

Table 1 continued

S. No.	PSO Variant	Year	Type of Parallelization + Strategy	Application	Objective Function
34	PPSO [54]	2010	MPI, Master-slave	Inverse heat conduction problem	Minimization of the error temperature value $w.r.t.$ aimed heat conduction parameters
35	PSO-MSAF [51]	2010	MPI, MATLAB parallel computing toolbox	To solve large-scale ED problem	3, 15 and 40-generator ED problems
36	Hardware PPSO [39]	2010	Multi-core, Master-slave	Five benchmark functions	For high-performance realization of PSO in embedded systems to speedup calculations
37	GSP PSO [40]	2011	MPI, Master-slave	Solving complex and large-scale problems by hiding communication latency	Three benchmark functions and TSP objective
38	Multi-kernel PPSO [121]	2011	GPU, Global, Random and Ring topology	To discuss possible approaches of parallelizing PSO	Three benchmark functions
39	Modified VEPSO [128]	2011	GPU	Implementation of Parallel MOPSO on consumer-level GPUs	Four double-objective test functions
40	PPSO [84]	2011	Multi-threading with four parallelization strategies	To evaluate the difference between the individual and simultaneous activity of particles	Three objectives: tracking a static and a dynamic target, then involving obstacle avoidance
41	MOPSO [129]	2011	GPU using CUDA and OpenCL, Master-slave	Comparison of MOPSO on GPU and OpenCL	Three benchmark functions
42	PPSO [112]	2011	GPGPU	To form low-cost architecture	Rastrigins and Ackleys functions on a 30-dimensional search space
43	PPSO-IR [55]	2011	Multi-core	CACM, RCV1 collections and randomly generated larger data sets	To obtain maximum similarity between the document and the query
44	pEPSO [103]	2011	CUDA, Fine-grained	To improve the processing speed and accuracy in high-dimensional problem	Five benchmark functions
45	CUDA PSO [115]	2012	GPU with multi-core parallelism	Geophysical Inversion	Maximization of speedup
46	Parallel CUDA-PSO [113]	2012	GPU	Minimize harmonic in multilevel inverters	Rosenbrock function
47	Parallel VEPSO [57]	2012	MPI, Peer-to-peer architecture	Design optimization of multi-objective problem of laminated composite plates	Minimization of weight and total cost of the composite laminated plates
48	PCOMPSO [42]	2012	MPI, Multi-core, Master-slave	Parallel implementation of MPSO-based cooperative approach	Five benchmark functions
49	LPSO [56]	2012	MATLAB parallel computing toolbox	To solve problem of static transmission expansion planning	Optimization of the expansion planning problem and handle the operational problem

Table 1 continued

S. No.	PSO Variant	Year	Type of Parallelization + Strategy	Application	Objective Function
50	MRCPSO [41]	2012	Hadoop MapReduce, Coarse-grained	Clustering of large-scale data	Minimization of the distance between all data points and particle centroids
51	PPSO_GM [127]	2012	GPU, Coarse-grained	To improve the convergence	Ten benchmark functions
52	Parallel GPSO [114]	2012	GPU	To solve sufficiently large and complex optimization problems	Optimization of radio frequency resource allocation
53	VNPSO [89]	2013	Multi-core with VC++	Scheduling problem for solar cell industry	Minimization of the makespan of the schedule
54	PPSO [86]	2013	MATLAB parallel computing toolbox	Real-time UAV path planning	Minimization of the cost function of the path composed of line segments, circular arcs and vertical helices
55	Parallel cooperative PSO [104]	2013	GPU	A comparison of serial and parallel implementation	Five benchmark functions
56	CP-PSO [122]	2013	CUDA	To analyze the impact of problem properties on the execution time	Five benchmark functions
57	Weighted Pareto PPSO [90]	2013	MATLAB parallel computing toolbox	Day-ahead V2G scheduling	Minimization of total operation costs and maximization of V2G income
58	PDPSO [105]	2013	GPU fine-grained parallelism	Massive parallelization of PSO algorithm onto a GPU-based architecture	Three benchmark functions
59	PPSO [88]	2013	MPI and Multi-core, Master-slave	PID controller tuning	Minimize error to achieve optimal performance
60	Agent-based PPSO [59]	2014	OpenMP, Master-slave	Obtaining solution of large-scale optimization problem on faster convergence rate	Five benchmark functions
61	TVPSO [58]	2014	Hadoop	Stock trading in financial market	Maximization of the annual net profit generated by PRS
62	PTVPSO-SVM [60]	2014	PVM	Parameter settings and feature selection of SVM	Maximization of fitness comprised of classification accuracy, number of selected features and the number of SVs
63	MsP-MmPSO [63]	2014	MATLAB parallel computing toolbox	Extracting fuzzy association rules	Fitness function with support and confidence parameters
64	Cellular PPSO [106]	2014	GPU CUDA, Fine-grained	Flexible job shop scheduling problem	Minimization of makespan
65	Parallel TVPSO [62]	2014	PVM, Master-slave	Simultaneously performing the parameter optimization and feature selection for SVM	Maximization of generalization capability of the SVM classifier

Table 1 continued

S. No.	PSO Variant	Year	Type of Parallelization + Strategy	Application	Objective Function
66	PCLPSO [43]	2015	MPI, Greedy information swap cooperation strategy	To solve large-size global optimization problems	Fourteen benchmark functions
67	MOPSO/D [98]	2015	OpenMP & MPI, Mix of master-slave and peer-to-peer approaches	To fully use the processing power of multi-core processors and a cluster	Minimization of inverted generational distance and Maximization of speed up
68	Parallel SASPSO [66]	2015	MPI	To solve conditional nonlinear optimal perturbation problem fast	Maximization of nonlinear evolution of the initial perturbation
69	PB-PPSO [61]	2015	CloudSim with virtual machine	To obtain the optimized resources in cloud for the set of tasks that have less makespan and minimum price	Minimization of makespan and total cost for the execution of the task
70	AIU-PSO [68]	2015	MPI, Master-slave	Large dimensional engineering problems	Parameter estimation objective function by ODRPACK95 optimizer [130]
71	Parallel MPSO [70]	2015	MPI, Master-slave with multi-core	Deformation control in earth-rockfill dam design	Deviation minimization and two benchmark functions
72	GPU-PSO, GPU-DPSO [116]	2015	GPU	Maximal constraint satisfaction problems	Minimization of the number of violated constraints
73	PCCPSO [91]	2015	MPI, Master-slave	Solving CNOP problem	Optimization of magnitudes and patterns of CNOP
74	CPPSO [123]	2015	GPU, Ring global topology	Complete resource use of the massively parallel multi-cores architectures	Four benchmark functions
75	PPSO [64]	2015	Apache Spark platform, Open-vSwitch	Learning analytics	Minimization of processing time, intra-cluster distance and inter-cluster distance
76	PPSO [107]	2015	GPU, Fine-grained + Hybrid	Optimization of PSO for the GPU instance of the AWS	Four benchmark test functions
77	PPSO [65]	2015	Master-slave, OpenMP	Reactive power and voltage control, along with investigating dependability	Minimization of active power losses
78	PPSO [44]	2015	MPI, Master-slave	Integrating the advantages of different computation paradigms	Thirteen benchmark functions
79	PPSO [72]	2015	OpenCL, Multi-core	To speedup the parameter estimation process for various PV models	Minimization of the root-mean-square error to minimize aggregate absolute difference into a single measure of predictive power

Table 1 continued

S. No.	PSO Variant	Year	Type of Parallelization + Strategy	Application	Objective Function
80	PPSO [67]	2015	MPI, Heterogeneous multiprocessor	Dynamic voltage scaling with heterogeneous multiprocessor system	Minimizing schedule length with energy consumption constraint and minimizing energy consumption with schedule length constraint
81	Parallel BPSO [69]	2015	MPI, Master-slave	Unsupervised change detection in multi-temporal multispectral satellite images	Fitness function with computational efficiency and accuracy
82	PPSO [71]	2015	OpenMP	Reactive power and voltage control	Minimization of active power losses
83	GPU SPSO [108]	2016	GPU, Fine-grained	Reduce execution time	Seven benchmark functions
84	MPSO [125]	2016	GPU with OpenCL, Ring topology	To improve solution quality and performance of hybrid multi-core machine	Twenty benchmark functions
85	PPSO-BP neural network algorithm [45]	2016	Hadoop	Image classification by BP neural network algorithm	Train the training samples to set weights and thresholds until it generates the expected output
86	PCPSO and kPSO [73]	2016	Hadoop	To optimize service composition in mobile environment	To optimize weighted root mean square of five QoS parameters
87	PPSO [126]	2016	GPU, MATLAB parallel computing toolbox	Obtain inverse of T1T2 spectrum by joint method	Minimization of noise
88	PPSO [74]	2016	MPI, Master-slave	Parameter identification and modeling of enzyme-catalytic time-delayed switched system	Minimization of biological robustness
89	PPSO_ring, PPSO_star, PPSO_share [94]	2016	Multi-core with ring, star and share topology	Operation of inter-basin water transfer-supply systems	Maximization of the water supply and minimization of water spillage of recipient reservoirs
90	PSA [75]	2016	Multi-core and GPU	Parameter estimation of photovoltaic cell model	Minimization of root-mean-square error for current value
91	PC_QPSO [46]	2016	MPI, Ring and fully connected topology	To decompose the high-dimensional problems into several sub-problems	Six benchmark functions
92	GPU-PSO [124]	2016	GPU	Trajectory optimization	Four benchmark functions
93	PNSQPSO [93]	2016	MPI, Peer to peer communication with star topology	Diversity enhancement of swarm and reduce premature convergence of QPSO	Five benchmark functions
94	PCCPSO [92]	2016	Hadoop Spark, Master-slave	To raise degree of parallelism, decrease computation time and improve algorithm efficiency	Seven benchmark functions

Table 1 continued

S. No.	PSO Variant	Year	Type of Parallelization + Strategy	Application	Objective Function
95	MOPSO and PSO with Gaussian mutation weight [99]	2016	Multi-core	Energy resource management	Maximization of profits and Minimization of the CO_2 emissions
96	FGP-PSO [47]	2017	OpenMP & MPI, Fine-grained	In computationally demanding optimization problem with very large number of dimensions in objective	Four benchmark functions
97	APSO [109]	2017	GPU with multi-core parallelism	Enhance the efficiency of PSO algorithm to solve high-dimensional and crucial global optimization problems	Two functions: 1. Composite Function, 2. Geostatic Correction
98	S-PGD, V-PGD [117]	2017	GPU	For various types of graph drawing	Minimization of the energy contribution of the link between two vertices
99	GPU PPSO [118]	2017	GPU, Master-slave	Hyper-parameter selection in DNNs	Minimization of Loss function
100	MPPSO [76]	2017	Multi-Core	To improve computational efficiency of long-term optimal hydro system operation	Maximization of the total energy production
101	PPSO [77]	2017	MATLAB parallel computing toolbox	Determine longevity, stability and economy of excavator for optimal design	Four optimization sub-objectives forming a single objective by normalization and weighting method
102	PCPSO [78]	2018	Multi-core	Seismic travel time tomography	Six benchmark functions
103	CCSMPSO [48]	2018	Multi-core	Computational speedups, higher convergence speed and solution quality	Benchmark problems of MOP
104	PPBO [49]	2018	Multi-core	To determine migration interval, rate, and direction	Thirteen benchmark functions
105	MAPSO1, MAPSO2 [79]	2018	Multi-core	A fast and efficient algorithm reconfigurable for an unpredictable environment-related embedded system	Three FJSP benchmark data set
106	PPSO, PGA [20]	2018	OpenMP, GPU	To evaluate FPGA-based parallel design and implementation method	Minimization of makespan
107	DEEPSO [80]	2018	OpenMP, Master-slave	Enhanced processing and improvement of solution quality in VQC	Minimization of active power loss in complete power system
108	DPSO [119]	2018	GPU, Hadoop	Power consumption minimization for luminance control	Minimization of power consumption with sufficient luminance

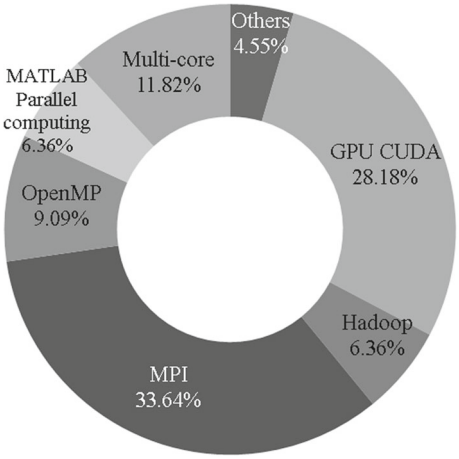

Fig. 3 Parallelization strategy-based publication analysis on Parallel PSO

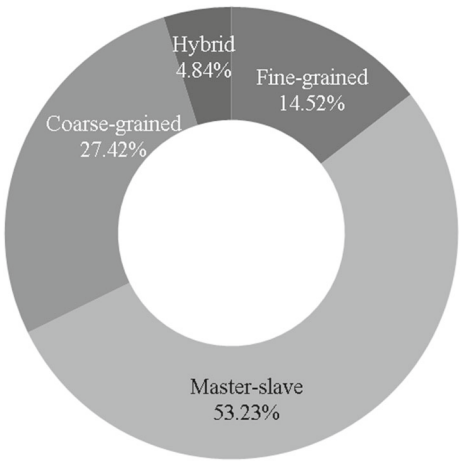

Fig. 4 Communication model-based publication analysis on Parallel PSO

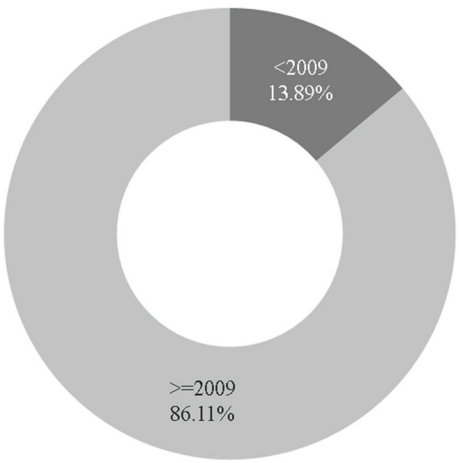

Fig. 5 Publication scenario in last decade

sive at cost criteria; hence, MPI-based parallelization is until now the most popular strategy. Similarly, master-slave topology is still the most popular communication topology due to its primitiveness. This article provides an overview of parallelization strategies available for researchers and their possible implementation pathways.

In future, the research on formulating the complex problems in form of multi-objective (or many-objective) optimization problem and then solving them with suitable parallelization strategy could be an advantageous research avenue. Similarly, for the problems with several variables or for big data problems, parallelization can effectively enhance the efficiency and performance of the implementation by employing parallel versions of heuristics.

Acknowledgements The first author (S.L.) gratefully acknowledges Science & Engineering Research Board, DST, Government of India, for the fellowship (PDF/2016/000008).

References

1. Bergh, V.: An Analysis of Particle Swarm Optimizers. Ph.D. thesis, Faculty of Natural and Agricultural Science, University of Pretoria (2001)
2. Kennedy, J.F.; Eberhart, R.C.: Particle swarm optimization. In: Proceedings of IEEE International Conference on Neural Networks, Piscataway, NJ, pp. 1942–1948 (1995)
3. Umbarkar, A.J.; Joshi, M.S.: Review of parallel genetic algorithm based on computing paradigm and diversity in search space. ICTACT J. Soft Comput. **3**(4), 615–622 (2013)
4. Cao, B.; Zhao, J.; Zhihan, L.; Liu, X.; Yang, S.; Kang, X.; Kang, K.: Distributed parallel particle swarm optimization for multi-objective and many-objective large-scale optimization. IEEE Access Spec. Sect. Big Data Anal. Internet Things Cyber-Phys. Syst. **5**, 8214–8221 (2017)
5. Lalwani, S.; Kumar, R.; Gupta, N.: A novel two-level particle swarm optimization approach to train the transformational grammar based hidden Markov models for performing structural alignment of pseudoknotted RNA. Swarm Evolut. Comput. **20**, 58–73 (2015)
6. Selvi, S.; Manimegalai, D.: Task scheduling using two-phase variable neighborhood search algorithm on heterogeneous computing and grid environments. Arab. J. Sci. Eng. **40**(3), 817–844 (2015)
7. Fernandez-Villaverdey, J.; Zarruk-Valenciaz, D.: A Practical Guide to Parallelization in Economics. University of Pennsylvania, Philadelphia (2018)
8. The Apache Software Foundation. Apache Hadoop. http://hadoop.apache.org/ (2018)
9. MATLAB and Simulink. https://in.mathworks.com/ (2018)
10. Wickham, H.: Advanced R. Chapman and Hall/CRC The R Series. Taylor and Francis, Milton Park (2014)
11. The Julia Language. https://docs.julialang.org/en/stable/manual/parallel-computing. Julia Parallel Computing (2018)
12. Gorelick, M.; Ozsvald, I.: High Performance Python: Practical Performant Programming for Humans. O'Reilly Media, Sebastopol (2014)
13. Bjarne Stroustrup.: Past, present and future of C++. http://cppcast.com/2017/05/bjarne-stroustrup/ (2017)
14. The OpenMP API specification for parallel programming. http://www.openmp.org/ (2018)

15. Gropp, W.; Lusk, E.; Skjellum, A.: Using MPI: Portable Parallel Programming with the Message-Passing Interface, vol. 1. MIT Press, Cambridge (1999)
16. nVIDIA.: nVIDIA CUDA Programming Guide v.2.3. nVIDIA Corporation, Santa Clara (2009)
17. Mei, G.; Tipper, J.C.; Xu, N.: A generic paradigm for accelerating laplacian-based mesh smoothing on the GPU. Arab. J. Sci. Eng. **39**(11), 7907–7921 (2014)
18. Farber, R.: Parallel Programming with OpenACC. Morgan Kaufmann, Burlington (2017)
19. Kaz, S.: An in-depth look at Google'first Tensor Processing Unit. https://cloud.google.com/blog/bigdata/2017/05/an-in-depth-look-at-googles-first-tensor-processing-unit-TPU (2018)
20. Zou, X.; Wang, L.; Tang, Y.; Liu, Y.; Zhan, S.; Tao, F.: Parallel design of intelligent optimization algorithm based on FPGA. Int. J. Adv. Manuf. Technol. **94**(9), 3399–3412 (2018)
21. Cantu-Paz, E.: Efficient and Accurate Parallel Genetic Algorithms. Kluwer Academic Publishers, Norwell (2000)
22. Madhuri, A., Deep, K.: A state-of-the-art review of population-based parallel meta-heuristics. In: World Congress on Nature and Biologically Inspired Computing, pp. 1604–1607 (2009)
23. Gies, D.; Rahmat-Samii, Y.: Reconfigurable array design using parallel particle swarm optimization. IEEE Int. Symp. Antennas Propag. Soc. **1**, 177–180 (2003)
24. Schutte, J.F.; Fregly, B.J.; Haftka, R.T.; George, A.D.: A parallel particle swarm optimizer. Technical report, Florida University, Gainesville Mechanical and Aerospace Engineering (2003)
25. Schutte, J.F.; Reinbolt, J.A.; Fregly, B.J.; Haftka, R.T.; George, A.D.: Parallel global optimization with the particle swarm algorithm. Int. J. Numer. Methods Eng. **61**(13), 2296–2315 (2004)
26. Cui, S.; Weile, D.S.: Application of a parallel particle swarm optimization scheme to the design of electromagnetic absorbers. IEEE Trans. Antennas Propag. **53**(11), 3616–3624 (2005)
27. Venter, G.; Sobieszczanski-Sobieski, J.: Parallel particle swarm optimization algorithm accelerated by asynchronous evaluations. J. Aerosp. Comput. Inf. Commun. **3**(3), 123–137 (2006)
28. Chusanapiputt, S.; Nualhong, D.; Jantarang, S.; Phoomvuthisarn, S.: Relative velocity updating in parallel particle swarm optimization based lagrangian relaxation for large-scale unit commitment problem. In: IEEE Region 10 Conference, Melbourne, Qld., Australia, pp. 1–6 (2005)
29. Koh, B.-I.; George, A.D.; Haftka, R.T.; Fregly, B.J.: Parallel asynchronous particle swarm optimization. Int. J. Numer. Methods Eng. **67**(4), 578–595 (2006)
30. McNabb, A.W.; Monson, C.K.; Seppi, K.D.: Parallel PSO using MapReduce. In: IEEE Congress on Evolutionary Computation, pp. 7–14 (2007)
31. Liu, Q.; Li, T.; Liu, Q.; Zhu, J.; Ding, X.; Wu, J.: Two phase parallel particle swarm algorithm based on regional and social study of object optimization. In: Third IEEE International Conference on Natural Computation, vol. 3, pp. 827–831 (2007)
32. Han, F.; Cui, W.; Wei, G.; Wu, S.: Application of parallel PSO algorithm to motion parameter estimation. In: 9th IEEE International Conference on Signal Processing, pp. 2493–2496 (2008)
33. Wang, D.; Wu, C.H.; Ip, A.; Wang, D.; Yan, Y.: Parallel multi-population particle swarm optimization algorithm for the uncapacitated facility location problem using openMP. In: IEEE World Congress on Computational Intelligence Evolutionary Computation, pp. 1214–1218 (2008)
34. Jeong, H.M.; Lee, H.S.; Park, J.H.: Application of parallel particle swarm optimization on power system state estimation. In: Transmission and Distribution Conference and Exposition: Asia and Pacific, pp. 1–4 (2009)
35. Lihua, C.; Yadong, M.; Na, Y.: Parallel particle swarm optimization algorithm and its application in the optimal operation of cascade reservoirs in Yalong river. In: Second IEEE International Conference on Intelligent Computation Technology and Automation vol. 1, pp. 279–282 (2009)
36. Kalivarapu, V.; Foo, J.L.; Winer, E.: Synchronous parallelization of particle swarm optimization with digital pheromones. Adv. Eng. Softw. **40**(10), 975–985 (2009)
37. Singhal, G.; Jain, A.; Patnaik, A.: Parallelization of particle swarm optimization using message passing interfaces (MPIs). In: IEEE World Congress on Nature and Biologically Inspired Computing, pp. 67-71 (2009)
38. Lorion, Y.; Bogon, T.; Timm, I.J.; Drobnik, O.: An agent based parallel particle swarm optimization-APPSO. In: IEEE Swarm Intelligence Symposium, pp. 52–59 (2009)
39. Farmahini-Farahani, A.; Vakili, S.; Fakhraie, S.M.; Safari, S.; Lucas, C.: Parallel scalable hardware implementation of asynchronous discrete particle swarm optimization. Eng. Appl. Artif. Intell. **23**(2), 177–187 (2010)
40. Li, B.; Wada, K.: Communication latency tolerant parallel algorithm for particle swarm optimization. Parallel Comput. **37**(1), 1–10 (2011)
41. Aljarah, I.; Ludwig, S.A.: Parallel particle swarm optimization clustering algorithm based on MapReduce methodology. In: Fourth IEEE World Congress on Nature and Biologically Inspired Computing, pp. 104–111 (2012)
42. Parsopoulos, K.E.: Parallel cooperative micro-particle swarm optimization: a master slave model. Appl. Soft Comput. **12**(11), 3552–3579 (2012)
43. Gulcu, S.; Kodaz, H.: A novel parallel multi-swarm algorithm based on comprehensive learning particle swarm optimization. Eng. Appl. Artif. Intell. **45**, 33–45 (2015)
44. Zhang, G.W.; Zhan, Z.H.; Du, K.J.; Lin, Y.; Chen, W.N.; Li, J.J.; Zhang, J.: Parallel particle swarm optimization using message passing interface. In: Proceedings of the 18th Asia Pacific Symposium on Intelligent and Evolutionary Systems, vol. 1, pp. 55–64 (2015)
45. Cao, J.; Cui, H.; Shi, H.; Jiao, L.: Big Data: a parallel particle swarm optimization-back-propagation neural network algorithm based on MapReduce. PLoS ONE **11**(6), e0157551 (2016)
46. Tian, N.; Wang, Y.; Ji, Z.: Parallel coevolution of quantum-behaved particle swarm optimization for high-dimensional problems. In: Asian Simulation Conference, pp. 367–376 (2016)
47. Nedjah, N.; Rogerio, M.C.; Luiza, M.M.: A fine-grained parallel particle swarm optimization on many core and multi-core architectures. In: International Conference on Parallel Computing Technologies, pp. 215–224 (2017)
48. Arash, A.; Bernabe, D.; Gregoire, D.; Pascal, B.: A scalable parallel cooperative coevolutionary PSO algorithm for multi-objective optimization. J. Parallel Distrib. Comput. **112**, 111–125 (2018)
49. Lai, X.; Zhou, Y.: An adaptive parallel particle swarm optimization for numerical optimization problems. Neural Comput. Appl. 1–19 (2018)
50. Li, Y.; Cao, Y.; Liu, Z.; Liu, Y.; Jiang, Q.: Dynamic optimal reactive power dispatch based on parallel particle swarm optimization algorithm. Comput. Math. Appl. **57**(11), 1835–1842 (2009)
51. Subbaraj, P.; Rengaraj, R.; Salivahanan, S.; Senthilkumar, T.R.: Parallel particle swarm optimization with modified stochastic acceleration factors for solving large scale economic dispatch problem. Int. J. Electr. Power Energy Syst. **32**(9), 1014–1023 (2010)
52. Li, Z.; Chen, Y.: Design and implementation for parallel particle swarm optimization color quantization algorithm. In: IEEE International Conference on Computer and Information Application, pp. 339–342 (2010)
53. Prasain, H.; Jha, G.K.; Thulasiraman, P.; Thulasiram, R.: A parallel particle swarm optimization algorithm for option pricing. In: IEEE International Symposium on Parallel and Distributed Processing, Workshops and PhD Forum (IPDPSW), pp. 1–7 (2010)

54. Qi, J.; Guo, Q.; Lin, J.; Zhou, M.; Zhang, S.: Parallel particle swarm optimization algorithm of inverse heat conduction problem. In: Ninth IEEE International Symposium on Distributed Computing and Applications to Business Engineering and Science, pp. 5-9 (2010)
55. Drias, H.: Parallel swarm optimization for web information retrieval. In: Third IEEE World Congress on Nature and Biologically Inspired Computing, pp. 249–254 (2011)
56. Torres, S.P.; Castro, C.A.: Parallel particle swarm optimization applied to the static transmission expansion planning problem. In: Sixth IEEE/PES Transmission and Distribution: Latin America Conference and Exposition, pp. 1–6 (2012)
57. Omkar, S.N.; Venkatesh, A.; Mudigere, M.: MPI-based parallel synchronous vector evaluated particle swarm optimization for multi-objective design optimization of composite structures. Eng. Appl. Artif. Intell. **25**(8), 1611–1627 (2012)
58. Wang, F.; Philip, L.H.; Cheung, D.W.: Combining technical trading rules using parallel particle swarm optimization based on Hadoop. In: IEEE International Joint Conference on Neural Networks, pp. 3987–3994 (2014)
59. Satapathy, A.; Satapathy, S.K.; Reza, M.: Agent-based parallel particle swarm optimization based on group collaboration. In: Annual IEEE India Conference, INDICON, pp. 1–5 (2014)
60. Xu, X.; Li, J.; Chen, H.l.: Enhanced support vector machine using parallel particle swarm optimization. In: 10th IEEE International Conference on Natural Computation, pp. 41–46 (2014)
61. Mohana, R.S.: A position balanced parallel particle swarm optimization method for resource allocation in cloud. Indian J. Sci. Technol. **8**(S3), 182–188 (2015)
62. Chen, H.L.; Yang, B.; Wang, S.J.; Wang, G.; Li, H.Z.; Liu, W.B.: Towards an optimal support vector machine classifier using a parallel particle swarm optimization strategy. Appl. Math. Comput. **239**, 180–197 (2014)
63. Gou, J.; Wang, F.; Luo, W.: Mining fuzzy association rules based on parallel particle swarm optimization algorithm. Intell. Autom. Soft Comput. **2**(2), 147–162 (2015)
64. Govindarajan, K.; Boulanger, D.; Kumar, V.S.: Parallel particle swarm optimization (PPSO) clustering for learning analytics. In: IEEE International Conference on Big Data, pp. 1461–1465 (2015)
65. Fukuyama, Y.: Parallel particle swarm optimization for reactive power and voltage control investigating dependability. In: 18th IEEE International Conference on Intelligent System Application to Power Systems, pp. 1–6 (2015)
66. Yuan, S.; Ji, F.; Yan, J.; Mu, B.: A parallel sensitive area selection-based particle swarm optimization algorithm for fast solving CNOP. In: International Conference on Neural Information Processing, pp. 71–78 (2015)
67. Kumar, P.R.; Babu, P.; Palani, S.: Particle swarm optimization based sequential and parallel tasks scheduling model for heterogeneous multiprocessor systems. Fundamenta Informaticae **139**(1), 43–65 (2015)
68. Moraes, A.O.S.; Mitre, J.F.; Lage, P.L.C.; Secchi, A.R.: A robust parallel algorithm of the particle swarm optimization method for large dimensional engineering problems. Appl. Math. Model. **39**(14), 4223–4241 (2015)
69. Kusetogullari, H.; Yavariabdi, A.; Celik, T.: Unsupervised change detection in multitemporal multispectral satellite images using parallel particle swarm optimization. IEEE J. Sel. Top. Appl. Earth Obs. Remote Sens. **8**(5), 2151–2164 (2015)
70. Jia, Y.; Chi, S.: Back-analysis of soil parameters of the Malutang II concrete face rockfill dam using parallel mutation particle swarm optimization. Comput. Geotech. **65**, 87–96 (2015)
71. Fukuyama, Y.: Verification of dependability on parallel particle swarm optimization based voltage and reactive power control. IFAC-PapersOnLine **48**(30), 167–172 (2015)
72. Ma, J.; Man, K.L.; Guan, S.; Ting, T.O.; Wong, P.W.H.: Parameter estimation of photovoltaic model via parallel particle swarm optimization algorithm. Int. J. Energy Res. **40**(3), 343–352 (2016)
73. Hossain, M.S.; Moniruzzaman, M.; Muhammad, G.; Ghoneim, A.; Alamri, A.: Big data-driven service composition using parallel clustered particle swarm optimization in mobile environment. IEEE Trans. Serv. Comput. **9**(5), 806–817 (2016)
74. Yuan, J.; Wang, L.; Xie, J.; Zhang, X.; Feng, E.; Yin, H.; Xiu, Z.: Modelling and parameter identification of a nonlinear enzyme-catalytic time-delayed switched system and its parallel optimization. Appl. Math. Model. **40**(19), 8276–8295 (2016)
75. Ting, T.O.; Ma, J.; Kim, K.S.; Huang, K.: Multicores and GPU utilization in parallel swarm algorithm for parameter estimation of photovoltaic cell model. Appl. Soft Comput. **40**, 58–63 (2016)
76. Sheng-li, L.; Ben-xi, L.; Chun-tian, C.; Zhi-fu, L.; Xin-yu, W.: Long-term generation scheduling of hydropower system using multi-core parallelization of particle swarm optimization. Water Resour. Manag. **31**(9), 1–17 (2017)
77. Xin, L.; Wang, G.; Miao, S.; Li, X.: Optimal design of a hydraulic excavator working device based on parallel particle swarm optimization. J. Braz. Soc. Mech. Sci. Eng., pp. 1–13 (2017)
78. Luu, K.; Noble, M.; Gesret, A.; Belayouni, N.; Roux, P.-F.: A parallel competitive particle swarm optimization for non-linear first arrival traveltime tomography and uncertainty quantification. Comput. Geosci. **113**, 81–93 (2018)
79. Nouiri, M.; Bekrar, A.; Jemai, A.; Niar, S.; Ammari, A.C.: An effective and distributed particle swarm optimization algorithm for flexible job-shop scheduling problem. J. Intell. Manuf. **29**(3), 603–615 (2018)
80. Yoshida, H.; Fukuyama, Y.: Parallel multi-population differential evolutionary particle swarm optimization for voltage and reactive power control in electric power systems. In: 56th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE), pp. 1240–1245 (2017)
81. Chu, S.C.; Pan, J.S.: Intelligent parallel particle swarm optimization algorithms. Parallel Evolut. Comput. **22**, 159–175 (2006)
82. Waintraub, M.; Schirru, R.; Pereira, C.: Multiprocessor modeling of parallel particle swarm optimization applied to nuclear engineering problems. Prog. Nucl. Energy **51**(6), 680–688 (2009)
83. Sivanandam, S.N.; Visalakshi, P.: Dynamic task scheduling with load balancing using parallel orthogonal particle swarm optimisation. Int. J. Bio-Inspir. Comput. **1**(4), 276–286 (2009)
84. Tu, K.Y.; Liang, Z.C.: Parallel computation models of particle swarm optimization implemented by multiple threads. Expert Syst. Appl. **38**(5), 5858–5866 (2011)
85. Zhang, Y.; Gallipoli, D.; Augarde, C.E.: Simulation based calibration of geotechnical parameters using parallel hybrid moving boundary particle swarm optimization. Comput. Geotech. **36**(4), 604–615 (2009)
86. Roberge, V.; Tarbouchi, M.; Gilles, L.: Comparison of parallel genetic algorithm and particle swarm optimization for real-time UAV path planning. IEEE Trans. Ind. Inform. **9**(1), 132–141 (2013)
87. Jin, N.; Rahmat-Samii, Y.: Parallel particle swarm optimization and finite-difference time-domain (PSO/FDTD) algorithm for multiband and wide-band patch antenna designs. IEEE Trans. Antennas Propag. **53**(11), 3459–3468 (2005)
88. Han, X.G.; Wang, F.; Fan, J.W.: The research of PID controller tuning based on parallel particle swarm optimization. Appl. Mech. Mater. **433**, 583–586 (2013)
89. Chen, Y.Y.; Cheng, C.Y.; Wang, L.C.; Chen, T.L.: A hybrid approach based on the variable neighborhood search and particle swarm optimization for parallel machine scheduling problems: a case study for solar cell industry. Int. J. Prod. Econ., **141**(1), 66–78 (2013)

90. Soares, J.; Vale, Z.; Canizes, B.; Morais, H.: Multi-objective parallel particle swarm optimization for day-ahead vehicle-to-grid scheduling. In: IEEE Symposium on Computational Intelligence Applications in Smart Grid, pp. 138–145 (2013)
91. Yuan, S.; Zhao, L.; Mu, B.: Parallel cooperative co-evolution based particle swarm optimization algorithm for solving conditional nonlinear optimal perturbation. In: International Conference on Neural Information Processing, pp. 87–95 (2015)
92. Cao, B.; Li, W.; Zhao, J.; Yang, S.; Kang, X.; Ling, Y.; Lv, Z.: Spark-based parallel cooperative co-evolution particle swarm optimization algorithm. In: IEEE International Conference on Web Services, pp. 570–577 (2016)
93. Long, H.X.; Li, M.Z.; Fu, H.Y.: Parallel quantum-behaved particle swarm optimization algorithm with neighborhood search. In: International Conference on Oriental Thinking and Fuzzy Logic, pp. 479–489 (2016)
94. Peng, Y.; Peng, A.; Zhang, X.; Zhou, H.; Zhang, L.; Wang, W.; Zhang, Z.: Multi-core parallel particle swarm optimization for the operation of inter-basin water transfer-supply systems. Water Resour. Manag. **31**(1), 27–41 (2017)
95. Vlachogiannis, J.G.; Lee, K.Y.: Determining generator contributions to transmission system using parallel vector evaluated particle swarm optimization. IEEE Trans. Power Syst. **20**(4), 1765–1774 (2005)
96. Fan, S.K.; Chang, J.M.: A parallel particle swarm optimization algorithm for multi-objective optimization problems. Eng. Optim. **41**(7), 673–697 (2009)
97. Vlachogiannis, J.G.; Lee, K.Y.: Multi-objective based on parallel vector evaluated particle swarm optimization for optimal steady-state performance of power systems. Expert Syst. Appl. **36**(8), 10802–10808 (2009)
98. Li, J-Z.; Chen, W-N.; Zhang, J.; Zhan, Z-H.: A parallel implementation of multiobjective particle swarm optimization algorithm based on decomposition. In: IEEE Symposium Series on Computational Intelligence, pp. 1310–1317 (2015)
99. Borges, N.; Soares, J.; Vale, Z.; Canizes, B.: Weighted sum approach using parallel particle swarm optimization to solve multi-objective energy scheduling. In: IEEE/PES Transmission and Distribution Conference and Exposition, pp. 1–5 (2016)
100. Li, J.; Wan, D.; Chi, Z.; Hu, X.: An efficient fine-grained parallel particle swarm optimization method based on GPU acceleration. Int. J. Innov. Comput. Inf. Control **3**(6), 1707–1714 (2007)
101. Zhou, Y.; Tan, Y.: GPU based parallel particle swarm optimization. In: IEEE Congress on Evolutionary Computation, Trondheim, Norway, pp. 1493–1500 (2009)
102. Hung, Y.; Wang, W.: Accelerating parallel particle swarm optimization via GPU. Optim. Methods Softw. **27**(1), 33–51 (2012)
103. Zhu, H.; Guo, Y.; Wu, J.; Gu, J.; Eguchi, K.: Paralleling Euclidean particle swarm optimization in CUDA. In: 4th IEEE International Conference on Intelligent Networks and Intelligent Systems, pp. 93–96 (2011)
104. Kumar, J.; Singh, L.; Paul, S.: GPU based parallel cooperative particle swarm optimization using C-CUDA: a case study. In: IEEE International Conference on Fuzzy Systems, Hyderabad, India, pp. 1–8 (2013)
105. Calazan, R.M.; Nedjah, N.; Luiza, M.M.: Parallel GPU-based implementation of high dimension particle swarm optimizations. In IEEE Fourth Latin American Symposium on Circuits and Systems, pp. 1–4 (2013)
106. Shenghui, L.; Shuli, Z.: Research on FJSP based on CUDA parallel cellular particle swarm optimization algorithm. In: International IET Conference on Software Intelligence Technologies and Applications, pp. 325–329 (2014)
107. Li, J.; Wang, W.; Hu, X.: Parallel particle swarm optimization algorithm based on CUDA in the AWS cloud. In: Ninth International Conference on Frontier of Computer Science and Technology, pp. 8–12 (2015)
108. Hussain, M.; Hattori, H.; Fujimoto, N.: A CUDA implementation of the standard particle swarm optimization. In: 18th IEEE International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC), Romania, pp. 219–226 (2016)
109. Wachowiak, M.P.; Timson, M.C.; DuVal, D.J.: Adaptive particle swarm optimization with heterogeneous multicore parallelism and GPU acceleration. IEEE Trans. Parallel Distrib. Syst. **28**(10), 2784–2793 (2017)
110. Chang, Y.L.; Fang, J.P.; Benediktsson, J.A.; Chang, L.; Ren, H.; Chen, K.S.: Band selection for hyperspectral images based on parallel particle swarm optimization schemes. IEEE Int. Geosci. Remote Sens. Symp. **5**, 84–87 (2009)
111. Mussi, L.; Cagnoni, S.; Daolio, F.: GPU based road sign detection using particle swarm optimization. In: Ninth IEEE International Conference on Intelligent Systems Design and Applications, Pisa, Italy, pp. 152–157 (2009)
112. Liera, I.C.; Liera, M.A.C.; Castro, M.C.J.: Parallel particle swarm optimization using GPGPU. In: CIE (2011)
113. Roberge, V.; Tarbouchi, M.: Efficient parallel particle swarm optimizers on GPU for real-time harmonic minimization in multilevel inverters. In: 38th Annual Conference on IEEE Industrial Electronics Society, pp. 2275–2282 (2012)
114. Rabinovich, M.; Kainga, P.; Johnson, D.; Shafer, B.; Lee, J.J.; Eberhart, R.: Particle swarm optimization on a GPU. In: IEEE International Conference on Electro/Information Technology, pp. 1–6 (2012)
115. Datta, D.; Mehta, S.; Srivastava, R.: CUDA based particle swarm optimization for geophysical inversion. In: 1st IEEE International Conference on Recent Advances in Information Technology, Dhanbad, India, pp. 416–420 (2012)
116. Dali, N.; Bouamama, S.: GPU-PSO: parallel particle swarm optimization approaches on graphical processing unit for constraint reasoning: case of Max-CSPs. Procedia Comput. Sci. **60**, 1070–1080 (2015)
117. Qu, J.; Liu, X.; Sun, M.; Qi, F.: GPU based parallel particle swarm optimization methods for graph drawing. Discrete Dyn. Nat. Soc., pp. 1–15 (2017)
118. Lorenzo, P.R.; Nalepa, J.; Ramos, L.S.; Pastor, J.R.: Hyper-parameter selection in deep neural networks using parallel particle swarm optimization. In: Proceedings of the Genetic and Evolutionary Computation Conference Companion, pp. 1864–1871 (2017)
119. Chih-Lun, L.; Shie-Jue, L.; Yu-Shu, C.; Ching-Ran, L.; Chie-Hong, L.: Power consumption minimization by distributive particle swarm optimization for luminance control and its parallel implementations. Expert Syst. Appl. **96**, 479–491 (2018)
120. Laguna-Sanchez, G.A.; Mauricio, O.C.; Nareli, C.C.; Ricardo, B.F.; Cedillo, J.: Comparative study of parallel variants for a particle swarm optimization algorithm implemented on a multithreading GPU. J. Appl. Res. Technol. **7**(3), 292–307 (2009)
121. Mussi, L.; Daolio, F.; Cagnoni, S.: Evaluation of parallel particle swarm optimization algorithms within the CUDA: a architecture. Inf. Sci. **181**(20), 4642–4657 (2011)
122. Altinoz, O.T.; Yilmaz, A.E.; Ciuprina, G.: Impact of problem dimension on the execution time of parallel particle swarm optimization implementation. In: 8th IEEE International Symposium on Advanced Topics in Electrical Engineering (ATEE), pp. 1–6 (2013)
123. Nedjah, N.; Calazan, R.M.; Luiza, M.M.; Wang, C.: Parallel implementations of the cooperative particle swarm optimization on many-core and multi-core architectures. Int. J. Parallel Program. **44**(6), 1173–1199 (2016)

124. Wu, Q.; Xiong, F.; Wang, F.; Xiong, Y.: Parallel particle swarm optimization on a graphics processing unit with application to trajectory optimization. Eng. Optim. **48**(10), 1679–1692 (2016)
125. Franz, W.; Thulasiraman, P.: A dynamic cooperative hybrid MPSO+GA on hybrid CPU+GPU fused multicore. In: IEEE Symposium Series on Computational Intelligence (SSCI), pp. 1–8 (2016)
126. Ge, X.; Wang, H.; Fan, Y.; Cao, Y.; Chen, H.; Huang, R.: Joint inversion of T1–T2 spectrum combining the iterative truncated singular value decomposition and the parallel particle swarm optimization algorithms. Comput. Phys. Commun. **198**, 59–70 (2016)
127. Jin, M.; Lu, H.: Parallel particle swarm optimization with genetic communication strategy and its implementation on GPU. In: IEEE 2nd International Conference on Cloud Computing and Intelligent Systems, vol. 1, pp. 99–104 (2012)
128. Zhou, Y.; Tan, Y.: GPU based parallel multi-objective particle swarm optimization. Int. J. Artif. Intell. **7**(A11), 125–141 (2011)
129. Arun, J.P.; Mishra, M.; Subramaniam, S.V.: Parallel implementation of MOPSO on GPU using OpenCL and CUDA. In: 18th IEEE International Conference on High Performance Computing, pp. 1–10 (2011)
130. Zwokak, J.W.; Boggs, P.T.; Watson, L.T.: ODRPACK95, Technical Report. Masters thesis, Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, Virginia, USA, (2004)