Hindawi
Complexity
Volume 2020, Article ID 8810759, 15 pages
https://doi.org/10.1155/2020/8810759

Research Article

Improving the Performance of Whale Optimization Algorithm through OpenCL-Based FPGA Accelerator

Qiangqiang Jiang,[1] Yuanjun Guo,[1] Zhile Yang,[1] Zheng Wang,[1] Dongsheng Yang,[2] and Xianyu Zhou[2]

[1]*Shenzhen Institutes of Advanced Technology, Chinese Academy of Sciences, Shenzhen, Guangdong 518055, China*
[2]*Intelligent Electrical Science and Technology Research Institute, Northeastern University, Shenyang 110819, China*

Correspondence should be addressed to Yuanjun Guo; yj.guo@siat.ac.cn

Received 25 August 2020; Revised 23 November 2020; Accepted 3 December 2020; Published 16 December 2020

Academic Editor: Shangce Gao

Whale optimization algorithm (WOA), known as a novel nature-inspired swarm optimization algorithm, demonstrates superiority in handling global continuous optimization problems. However, its performance deteriorates when applied to large-scale complex problems due to rapidly increasing execution time required for huge computational tasks. Based on interactions within the population, WOA is naturally amenable to parallelism, prompting an effective approach to mitigate the drawbacks of sequential WOA. In this paper, field programmable gate array (FPGA) is used as an accelerator, of which the high-level synthesis utilizes open computing language (OpenCL) as a general programming paradigm for heterogeneous System-on-Chip. With above platform, a novel parallel framework of WOA named PWOA is presented. The proposed framework comprises two feasible parallel models called partial parallel and all-FPGA parallel, respectively. Experiments are conducted by performing WOA on CPU and PWOA on OpenCL-based FPGA heterogeneous platform, to solve ten well-known benchmark functions. Meanwhile, other two classic algorithms including particle swarm optimization (PSO) and competitive swarm optimizer (CSO) are adopted for comparison. Numerical results show that the proposed approach achieves a promising computational performance coupled with efficient optimization on relatively large-scale complex problems.

1. Introduction

Swarm optimization or evolutionary algorithms have demonstrated their significance in a wide range of scientific and practical problems [1–5]. Recent years, more and more research studies' focus has been on multiobjective problems and artificial intelligence [6–9]. Whale optimization algorithm (WOA), a novel swarm intelligence-based metaheuristic algorithm, was proposed by Mirjalili and Lewis in 2016 [10]. Inspired by the special hunting behavior of humpback whales, WOA shows better performance compared with several existing popular methods and has drawn great research attention. Typically, Abdel-Basset et al. [11] integrated WOA with a locals search strategy for tackling the permutation flow shop scheduling problem. Mafarja and Mirjalili [12] proposed a hybrid WOA with simulated annealing for feature extraction. Aljarah et al. [13] introduced WOA-based trainer to train multilayer perceptron (MLP) neural networks. Moreover, there are also research bodies trying to tackle other diverse problems using WOA, such as multiobjective optimization [14–16], image processing [17–19], software testing [20], and power system applications [21, 22].

However, large-scale, multiple constraints and complex scenarios usually appear in actual engineering optimization problems, such as job shop scheduling, mixed unit commitment problem, and automatic path planning. Furthermore, high requirements in response speed and real-time performance need to be satisfied when solving problems above. In this situation, most optimization algorithms including WOA might get stuck in the executing dilemma. As the scale and complexity of the problem increase, the

execution time of WOA will increase rapidly, which leads to time-performance deterioration [23]. With inherent parallelism of WOA, the abovementioned problem can be tackled by applying parallel algorithm developed targeting specific accelerating platforms. In recent years, experts and scholars have tried to implement various swarm optimization algorithms employing state-of-the-art technologies such as multicore (message passing interface-MPI, OpenMP), distributed (MapReduce, Spark), and heterogeneous computing-based parallel platforms (graphics processing unit-GPU, FPGA).

Heterogeneous computing refers to using dedicated hardware devices with different architectures to execute time-consuming tasks, balancing the computational load of CPU. GPU is a classical parallel computing device, widely used in graphics visualization, image/video processing, scientific computing, deep learning, and so on. Nevertheless, with the increasing deployment of GPU, energy consumption and heat dissipation have become severe limitations for system extension, as well as brings heavy environmental pressure to human society [24]. In light of this, some researchers begin to choose other hardware devices to alleviate pressure caused by GPU. FPGA, a novel parallel accelerator, possesses powerful parallel computing capability and flexible programmability, while maintaining the advantage of low power consumption [25]. Traditional FPGA design, however, has drawbacks of high development difficulty and time consumption. Recently, Intel provides a development kit for software users, making it possible to deploy OpenCL program on FPGA. Consequently, developers can rapidly implement FPGA-based heterogeneous applications through OpenCL API thus reducing the development cost and time-to-market.

This research proposes high-performance parallel WOA (PWOA), with implementation on FPGA to effectively solve large-scale and complex optimization problems. More specifically, the main contributions of this paper are presented as follows:

(1) A novel heterogeneous parallel framework of WOA based on OpenCL-based FPGA accelerator.

(2) Two efficient models including partial parallel model and all-FPGA parallel model, with program flow design and dataflow analysis.

(3) Several diverse numerical experiments are conducted with ten selected benchmark functions. By comparing with sequential WOA executing on CPU, the proposed PWOA based on two parallel models achieves higher execution performance.

The rest of this paper is organized as follows: Section 2 represents a substantial literature review on exploration for parallel optimization algorithms. The theory of WOA and OpenCL-based FPGA heterogeneous accelerating platform is introduced in Section 3. Section 4 describes FPGA implementation of the proposed PWOAs with two parallel models and followed by the experimental results and statistical analysis in Section 5. Finally, conclusions are given in Section 6.

2. Related Work

Swarm optimization algorithms including WOA encounter challenges that the optimization performance decreases due to extensive computational cost when solving problems with high-dimension and complex mathematical model. To overcome these challenges, researchers have designed parallel swarm algorithms with implementation on various platforms. In recent years, distributed and parallel particle swarm optimization (PSO) has been implemented. Some studies [26–30] applied GPU to parallelize PSO, putting forward diverse parallel strategies. Hajewski and Oliveira [31] developed fast cache-aware parallel PSO relying on OpenMP. Ant colony optimization (ACO) [32] and artificial bee colony (ABC) [33] were also parallelized by GPU. Concerning brain storm optimization (BSO), Jin and Qin [34] presented GPU-based manner whilst Ma et al. [35] proposed parallelized BSO algorithm based on Spark framework for association rule mining. Similar works in [36, 37] used GPU and FPGA to accelerate genetic algorithm (GA). What deserves attention is that Garcia et al. [38] achieved parallel implementation and comparison of teaching-learning based optimization (TLBO) and Jaya on many-core GPU. As for WOA, Khalil et al. [39] proposed a simple and robust distributed WOA using Hadoop MapReduce, reaching a promising speedup.

It can be concluded that there are several typical kinds of parallel techniques including OpenMP, MapReduce, Spark, and heterogeneous architecture based on dedicated accelerators, such as GPU and FPGA. GPU becomes popular for general-purpose parallel computing as developing parallel swarm intelligence algorithms via GPU has successfully gained remarkable performance improvement [40]. Recently, FPGA is gradually applied to heterogeneous computing and algorithms accelerating based on OpenCL, which benefits from its high-parallelism, better energy efficiency, and flexible programmability [41–44]. The experiments conducted by [45] showed that swarm algorithms on FPGAs achieved a better speedup than that on GPUs and multicore CPUs. Nevertheless, designing a near-optimal accelerator is not an easy task. Implementing CPU-oriented codes on FPGA rarely increases the performance and even reduces the performance compared to CPU. Therefore, it requires not only the digital design expertise but also software skills to form appropriate OpenCL codes [46].

Few research works have been investigated on FPGA implementation of swarm optimization algorithms, especially WOA. Our prior work [47] explored WOA based on partial parallel scheme and deployed it on the FPGA heterogeneous platform. Then, empirical results using classic benchmarks proved the consequential advance of the

proposed methology in execution performance and convergence speed. In this paper, motivated by the previous studies, a novel PWOA scenario with two parallel models encompassed is further exploited via FPGA. Meanwhile, more diverse benchmarks are used to verify the effectiveness of PWOA based on FPGA parallel framework and its computational performance for large-scale complex problems.

3. Whale Optimization Algorithm and Acceleration Platform

3.1. Basic WOA Algorithm. The WOA algorithm constitutes two main phases, exploitation and exploration, through emulating shrinking encircling, bubble-net attacking, and searching for preys. The following subsections explain in detail the mathematical models of each phase.

3.1.1. Exploitation Phase (Encircling and Bubble-Net Attacking). To hunt preys, humpback whales first recognize the location of preys and encircle them. The mathematical model of shrinking encircling is represented by the following equations:

$$\mathbf{D} = \left| C \cdot \mathbf{X}^*_{(t)} - \mathbf{X}_{(t)} \right|, \tag{1}$$

$$\mathbf{X}_{(t+1)} = \mathbf{X}^*_{(t)} - A \cdot \mathbf{D}, \tag{2}$$

where $\mathbf{X}$ is the position vector, $\mathbf{X}^*$ represents the position of the best solution obtained so far, t indicates the current number of iteration, $||$ denotes the absolute operation, and $\cdot$ means an element-by-element multiplication. A and C are two parameters, which are calculated as follows:

$$A = 2a \cdot r - a, \tag{3}$$

$$C = 2 \cdot r, \tag{4}$$

where a is linearly decreasing from 2 to 0 through iterations (in both exploitation and exploration phases) and r is a random number in [0, 1]. The value of a is calculated by $a = 2 - t\,(2/\text{MaxIter})$, and MaxIter is the maximum number of iterations.

Another method used in the exploitation phase is spiral updating position, which in coordination with aforementioned shrinking encircling constitutes the bubble-net attacking strategy of humpback whales. The mathematical equations are as follows:

$$D' = \left| \mathbf{X}^*_{(t)} - \mathbf{X}_{(t)} \right|, \tag{5}$$

$$\mathbf{X}_{(t+1)} = D' \cdot e^{bl} \cdot \cos(2\pi l) + \mathbf{X}^*_{(t)}, \tag{6}$$

where b is a constant for determining the shape of the logarithmic spiral and l is a random number in $[-1, 1]$. Shrinking encircling and spiral updating position are used simultaneously during exploitation phase. The mathematical model is as follows:

$$\mathbf{X}_{(t+1)} = \begin{cases} \mathbf{X}^*_{(t)} - A \cdot \mathbf{D}, & p < 0.5, \\ D' \cdot e^{bl} \cdot \cos(2\pi l) + \mathbf{X}^*_{(t)}, & p \geq 0.5, \end{cases} \tag{7}$$

where p is a random value in $[-1, 1]$ which stands for a probability of 50% to choose either the shrinking encircling method or the spiral-shaped mechanism to update the position of whales during optimization process.

3.1.2. Exploration Phase (Searching for Preys). In addition to exploitation phase, a stochastic searching technique is also adopted to enhance the exploration in WOA. Unlike exploitation, a random whale $\mathbf{X}_{\text{rand}}$ is selected from swarm to navigate the search space, so as to find a better optimal solution (prey) than the existing one. This phase can efficiently prevent the algorithm from falling into local optima stagnation. Subsequently, based on the parameter A, a decision is made on which mechanism to be used for updating the position of whales. Exploration is done if $|A| \geq 1$, meanwhile if $|A| < 1$. The optimization process is mathematically described as follows:

$$\mathbf{D} = \left| C \cdot \mathbf{X}_{\text{rand}} - \mathbf{X}_{(t)} \right|, \tag{8}$$

$$\mathbf{X}_{(t+1)} = \mathbf{X}_{\text{rand}} - A \cdot \mathbf{D}, \tag{9}$$

where $\mathbf{X}_{\text{rand}}$ is a random position of the whale chosen from the current population and C is calculated by equation (4).

Algorithm 1 presents the pseudocode of WOA. At the beginning of the algorithm, an initial random population is generated, and each individual gets evaluated by fitness function and X^* is the current best solution. Then, the algorithm is repeatedly executed until the stop condition is satisfied. At each iteration, search agents update their position according to either a random chosen individual when $|A| \geq 1$, or the optimal solution obtained so far when $|A| < 1$. Depending on p, the WOA algorithm decides on whether using circular or spiral movement.

3.2. OpenCL-Based FPGA Heterogeneous Computing Platform

3.2.1. OpenCL and FPGA. OpenCL, maintained by Khronos Group, is an open standard for general-purpose parallel computing [48]. Various hardware devices, such as CPU, FPGA, GPU, and DSP, are supported for implementing highly efficient and parallel algorithms across heterogeneous computing platform. Additionally, OpenCL specifies a *C*99-based programming API for convenience of software developers. A typical OpenCL program consists of host and kernel sections.

FPGA is a configurable integrated circuit that can be repeatedly reconfigured to perform a huge number of logic functions. It generally includes programmable core logics, hierarchical reconfigurable interconnects, I/O elements, memory blocks, and DSPs. With these substantial logical resources, FPGA achieves an increased programming flexibility compared to application-specific integrated circuits

```
Generate initial population X_i (i = 1, 2, ..., n)
Evaluate the fitness of each search agent
X* = the best search agent
while (t < MaxIter) do
  for each search agent do
    Update a, A, C, l and p
    if (p < 0.5) then
      if (|A| < 1) then
        Update the position of the current search agent by equation (2)
      else if (|A| ≥ 1) then
        Select a random search agent (X_rand)
        Update the position of the current search agent by equation (9)
      end if
    else if (p ≥ 0.5) then
      Update the position of the current search agent by equation (6)
    end if
  end for
  Amend search agents which go beyond the search space
  Calculate the fitness of each search agent
  Replace X* with a better solution (if found)
  t = t + 1
end while
return X*
```

Algorithm 1:

(ASICs). However, traditional development flow on FPGA heavily relies on register transfer level (RTL) descriptions such as Verilog and very high speed integrated circuit hardware description language (VHDL), which incurs high development and verification cost. To address this problem, FPGA vendors such as Intel and Xilinx released OpenCL-based development flow which eases software developers to design FPGA-based applications, making this process more efficient.

3.2.2. Intel FPGA SDK for OpenCL. The Intel FPGA SDK for OpenCL [49] entitles developers to create high-level FPGA implementation with OpenCL. This SDK generates a heterogeneous computing environment where OpenCL kernels are compiled by Altera Offline Compiler (AOC) for programming FPGA at runtime. In this paradigm, Intel achieves design optimization while hiding low-level hardware details of FPGA. Subsequently, FPGA has gradually been applied to a wide range of fields such as image and video processing [42, 50], deep learning [51–53], and intelligent optimization algorithm [46].

OpenCL-based FPGA logic framework is illustrated in Figure 1 where several modules are specifically explained as follows:

(1)Kernel pipeline: the core module of entire framework, which is an implementation of specific functions. The kernel code is compiled by AOC offline compiler and will be synthesized into highly parallel optimized logic circuit referring to the internal architecture of FPGA.

(2)Processor: a host processor, typically CPU, used to control programs running on FPGA device.

(3)DDR: off-chip memory, including global and constant memory in the OpenCL memory model. Intel Cyclone V FPGA device used in this context has a DDR3 with a capacity of 1 GB. By default, the constant cache size is 16 KB and can be modified in accordance with practical requirements.

(4)PCI-e: high-speed data exchanging interface, responsible for transporting data and instruction between host and device.

(5)On-chip memory: internal memory of FPGA device, equivalent to local and private memory in the OpenCL memory model. With small capacity but high speed, it is mainly used for storing input and output temporary data, reducing the number of accesses to global memory. Thus, we may take advantage of on-chip memory to improve the efficiency of OpenCL program.

(6)Local memory interconnect: a bridge between executing unit and memory.

(7)External memory controller and PHY: a controller which is in charge of controlling data sending and receiving via DDR.

4. Parallel Whale Optimization Algorithm Based on FPGA

With the descriptions and definitions above, the framework of WOA can be summarized as shown in the left flowchart in Figure 2. Note, meanwhile, that the right flowchart is a simplified framework of WOA which is mainly composed of initialization, swarm updating, fitness calculation, and swarm evaluation. Similar to other swarm optimization

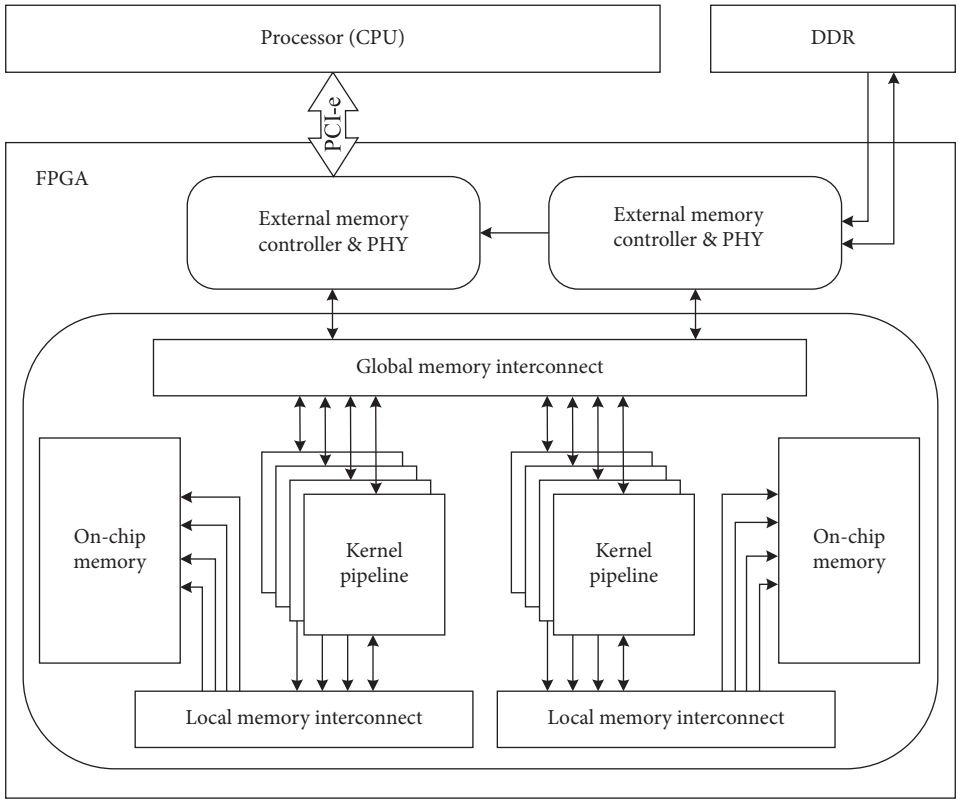

Figure 1: Architecture of heterogeneous platform with CPU and FPGA.

algorithms, WOA unavoidably suffers from this drawback of time-consuming operations such as updating swarm and calculating fitness, which greatly limits its execution speed [45]. Thanks to natural parallelism, the components utilized to implement swarm updating and fitness calculation in WOA can be executed concurrently. Within swarm updating phase, the positions of searching whales are updated separately by corresponding moving mechanism, more biologically simulating a real hunting process. For the remaining two phases, the initialization keeps primary ideology in this work for it has little effect on computational performance, whereas the evaluation is synchronous and cannot be paralleled.

This section will propose parallel WOA based on FPGA heterogeneous computing platform. In order to reach efficient acceleration, some compute-intensive tasks of WOA need to be transferred to FPGA side for parallel execution whilst CPU performs the remaining tasks. The parallel model can be divided into partial parallel and all-FPGA parallel by assigning different tasks to CPU and FPGA. Below is the PWOA implementation, which is described in two aspects: program flow design and dataflow analysis.

4.1. Initialization. Initialization mainly prepares basic data needed during the whole phase of WOA, including generating random numbers and initial population. This process is carried out at the beginning stage of WOA and executed only once. On top of this, C/C++ dedicated library for random number generation is applied as OpenCL does not support native random number generator. In this paper, a generic methodology, putting computational task of initialization at CPU side, is adopted into these two proposed parallel models, so as to take full use of computational horsepower from CPU.

Random number generation is a crucial component for WOA. On the one hand, the initial population is composed of whales with a random position. What needs to be ensured is that the value of the random position must be within the range of decision variable according to specific objective functions. On the other hand, there are several random numbers used as coefficients (a, A, C, l, and p) for updating the position of whale, which plays a significant role in optimization performance. Besides, these coefficients appear in each iteration, meaning that data transportation between FPGA and CPU also appears in each iteration. It will become a bottleneck for the running speed of PWOA due to frequent data transportation between FPGA and CPU. To alleviate this drawback, all random numbers required as well as the initial population are generated at CPU side and then sent to FPGA side once via OpenCL global memory. The proposed approach can substantially reduce the time overhead of PWOA.

4.2. Partial Parallel Model-Based PWOA

4.2.1. Program Flow Design. The partial parallel model executes several algorithmic sections in parallel involving the so-called master-slaves model. The partial parallel model-based PWOA (PWOA-PPM) on FPGA is presented in Figure 3.

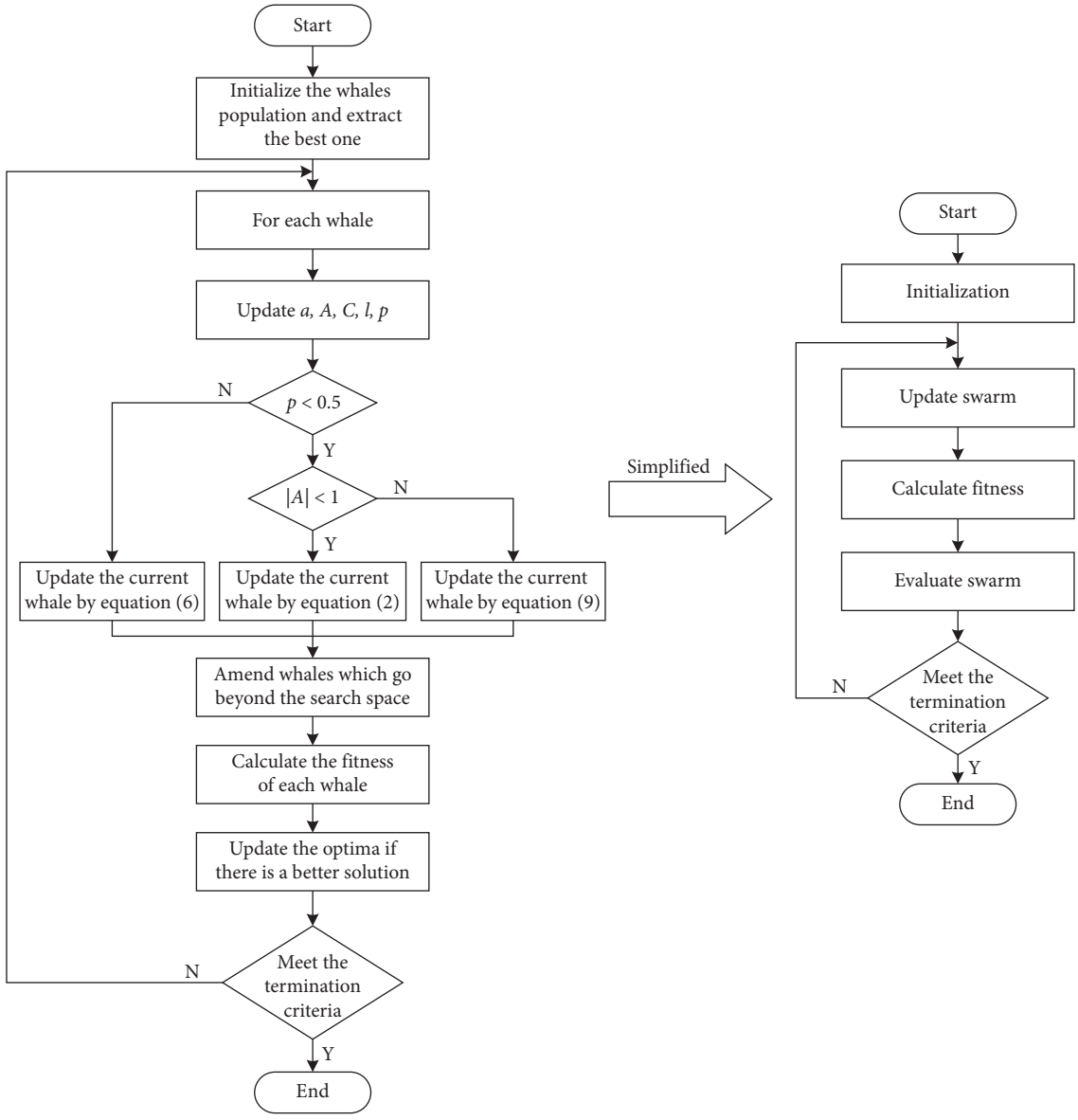

Figure 2: Framework of the WOA algorithm.

Host Program Flow. In terms of host program running on CPU, it undertakes the initialization of PWOA and transfers basic data related to kernel side via OpenCL global memory. Due to the restriction of synchronization, swarm evaluation is put on CPU for sequential executing in this model. After that, host program maintains the basic framework of WOA where it allocates tasks to FPGA, reads computation results from FPGA, and evaluates swarm in each iteration. The evaluation result is also sent to FPGA when host program enqueues task commands which drive kernel function to be executed on FPGA. Such task allocation can make better use of the processing power of CPU but correspondingly cause supernumerary communication overhead between CPU and FPGA.

Kernel Program Flow. FPGA device is used to deploy kernel program and accelerate it. Host offloads computationally intensive tasks onto FPGA for parallel computing. Based on the OpenCL programming model, the parallel parts of the algorithm are mapped to kernel function to be executed by threads (or work items) independently [40, 45]. In the proposed model, a fine-grained strategy is adopted, where each thread takes charge of an individual, calculating fitness and updating position. According to the coefficients (A and p), each thread (individual) performs different mechanisms simultaneously: shrinking encircling, spiral updating, or stochastic searching. Once kernel program finishes executing, the final results are written back to global memory.

4.2.2. Dataflow Analysis between Host and Kernel. In the proposed implementation, the dataflow between host and kernel mostly depends on global memory bandwidth. At host side, the memory buffers are created and the data used are mapped to these buffers, which will be further sent to the

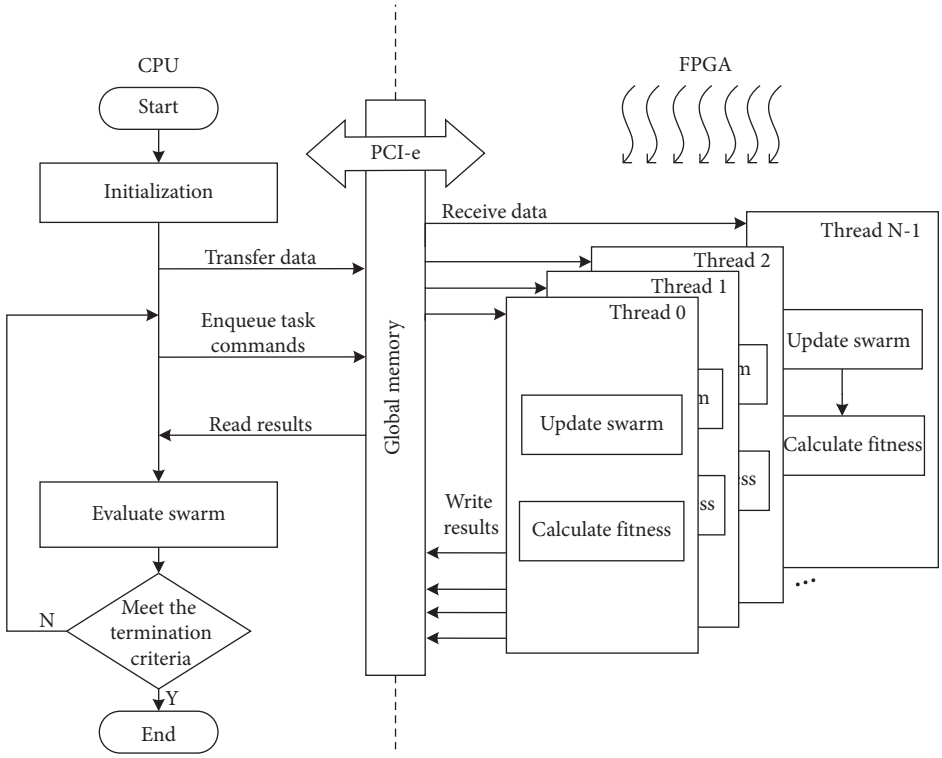

Figure 3: Framework of the partial parallel model-based PWOA (PWOA-PPM).

global memory of kernel via PCI-e. At kernel side, each thread works as a basic processing element, reads data from global memory and complete kernel function. As illustrated in Figure 4, the data set contains positions and fitnesses of all search agents, the global optima X^*, and coefficients (a, A, C, l and p). One block in the "positions" memory block and "fitnesses" memory block represents multi-dimension position information and fitness value of one whale individual, respectively. In the "coefficients" memory block, all coefficients required for a whale during the whole iterations are stored in one block, whereas the "optima" memory block just holds the position of the global best whale.

4.3. All-FPGA Parallel Model Based PWOA

4.3.1. Program Flow Design. In the all-FPGA parallel model, most constituent parts of WOA, except for initialization, are ported to FPGA. The All-FPGA parallel model-based PWOA (PWOA-AFPM) is designed as shown in Figure 5.

Host Program Flow. At host side (CPU), similar to the previous partial parallel model, host program undertakes the initialization of WOA and the basic data related are off-loaded onto kernel side via OpenCL global memory. However, it no longer controls the basic framework of WOA in this parallel model, making a relatively low workload for CPU while a greater computation overhead for FPGA. After completing the above two operations, the host program enqueues task commands to start the kernel program of FPGA and finally reads results from global memory. A dramatic advantage of this design, in comparison with partial parallel model, is minimal communication overhead between CPU and FPGA.

Kernel Program Flow. Within this model, kernel program running on FPGA becomes more complex than the previous model. In addition to receiving data and writing results back to global memory, the evolutionary framework, which contains swarm updating, fitness calculation, and swarm evaluation, is controlled by the kernel. Similarly, the fine-grained model is also applied to make multiple threads executing kernel function in parallel. Nevertheless, cares should be taken when evaluating swarm because all threads share one global optimal solution. To ensure the accuracy of the algorithm, we define memory consistency across threads with respect to memory fences [54, 55]. As depicted in Figure 5, the process in red dotted line is performed as synchronizer, where not only do all threads reach a synchronized state before this process, but also using a better solution obtained by any thread to substitute for the global optimal solution is an atomic operation as well. In this way, all threads can be executed in order, therefore guarantees the evaluation results.

4.3.2. Dataflow Analysis between Host and Kernel. In this model, the dataflow between host and kernel involves global memory and on-chip memory (local memory), which is presented in Figure 6. The same as the previous model, positions and coefficients are transmitted via global memory. To transmit the final result from the kernel to host, it also requires global memory to store this variable. Therefore,

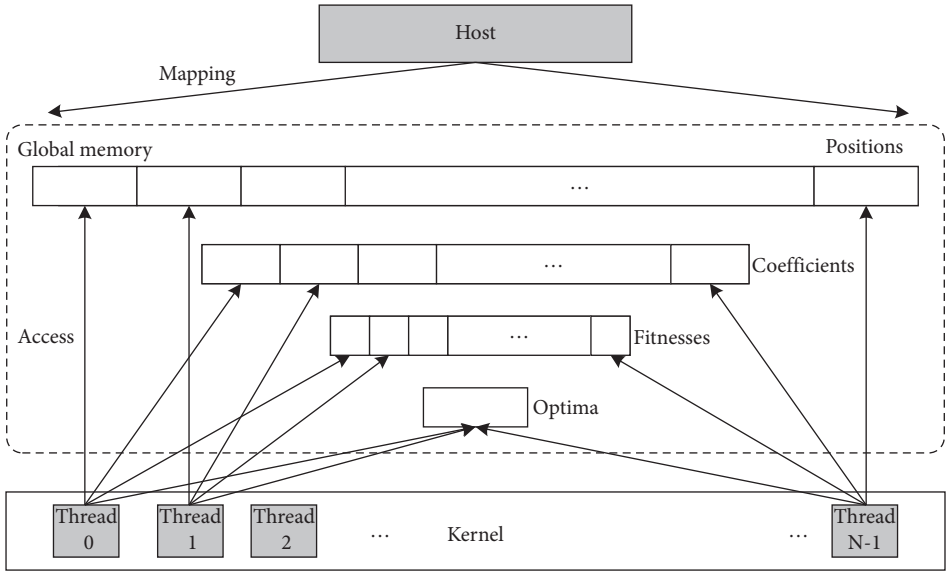

Figure 4: Dataflow of PWOA-PPM between host and kernel.

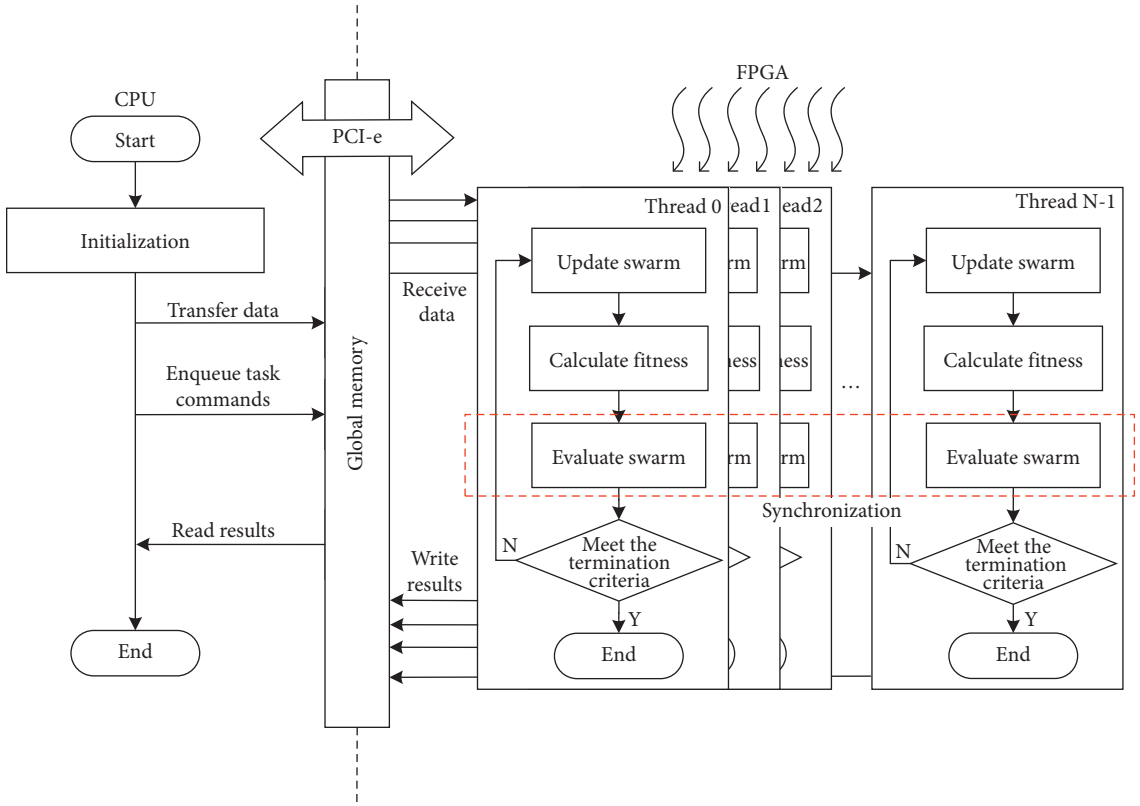

Figure 5: Framework of the All-FPGA parallel model-based PWOA (PWOA-AFPM).

a memory buffer is created by host program at the beginning, requesting a global memory space for the global optima X^*. Usage of on-chip memory from FPGA is a noticeable variation of dataflow between this model and the prior model illustrated in Figure 4. This is because most operations of WOA are executed by FPGA, and it is a rational strategy to utilize on-chip memory comprised of local memory and private memory. Besides, this kind of memory can be directly and efficiently requested during the process of executing kernel. Thus, the intermediate results, such as optima and the fitness of all individuals, are stored into local memory. Furthermore, a more efficient synchronous evaluation process also benefits from the dataset in local memory.

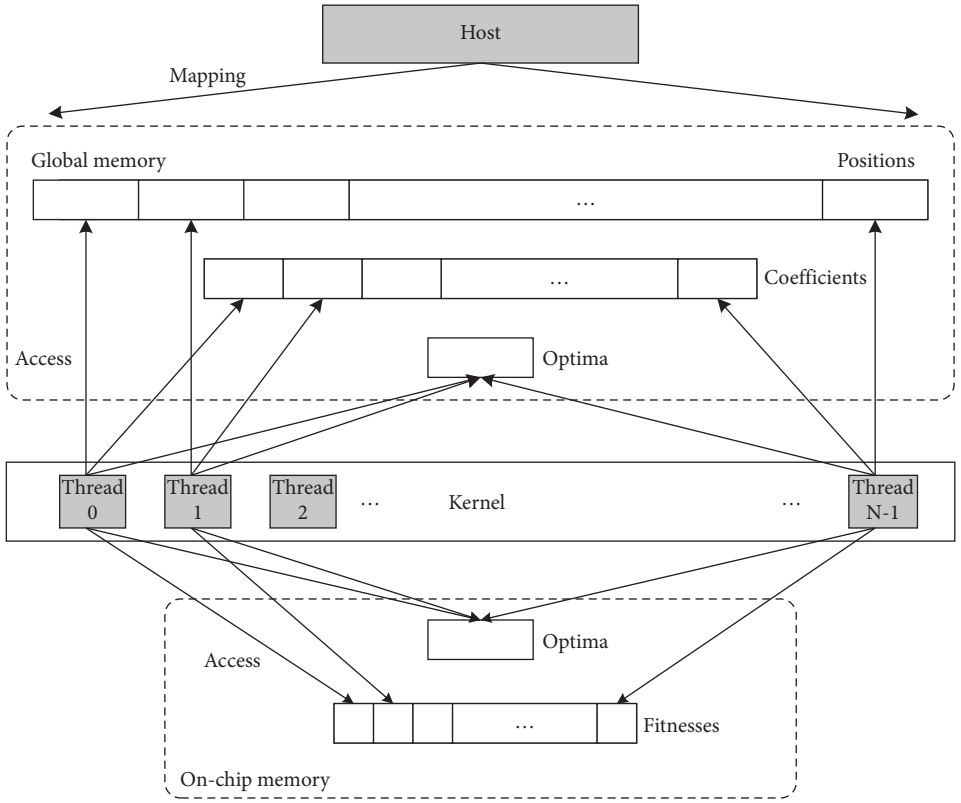

Figure 6: Dataflow of PWOA-AFPM between host and kernel.

5. Numerical Experiment and Analysis

5.1. Experimental Setup. The experimental platform contains two main hardware devices: CPU and FPGA. For CPU platform, Intel Core i5-8250 CPU with 16 GB RAM is used, while for FPGA platform, Intel FPGA Cyclone VGT with 1 GB DDR3 and 64 MB SDRAM is used. The entire development environment is based on Ubuntu 14.04 LTS and Intel FPGA SDK for OpenCL 17.1 version.

In this paper, ten general benchmark functions [56], listed in Table 1, are used to make performance comparisons between serial WOA (CPU implementation) and two parallel models based PWOA (FPGA implementation). Among these benchmark functions, $f1 \sim f5$ are unimodal functions while $f6 \sim f10$ are multimodal functions.

Concerning other parameters in the canonical WOA algorithm, coefficient b in the spiral-updating model is held constant during the whole evaluation process and set to be 1.0. Dimensions including 64 D, 128 D, 256 D, and 512 D are set for the optimization test, and the population size of WOA is dynamically set to be twice the size of the dimension. To verify the performance of the proposed PWOAs, other two canonical algorithms, PSO [57] and competitive swarm optimizer (CSO) [58], are selected for comparison. Additionally, for each implementation with a specific dimension setting, 30 independent runs are executed and the average performance is considered. For each independent run, the maximum number of fitness evaluations (FEs) is set to $1000 \times D$, where D is the search dimension of the test functions.

5.2. Optimization Result and Running Time on Benchmark Functions. By using three WOAs with different schemes and two state-of-the-art algorithms to optimize 10 benchmark functions, experimental data can be obtained as listed in Table 2–5, where *mean* and *time* refer to the average values of optimization result and running time for 30 runs.

Based on numerical values given in above tables, it can be noticed that WOA and PWOAs constructed by two parallel models (PWOA-PPM and PWOA-AFPM) present higher problem solving efficacy than CSO and PSO when optimizing all benchmark functions with several dimensions. As for mean results of all the 10 test cases, WOA and the proposed PWOAs obtain more accurate values, compared with other two algorithms. When optimizing $f1$, $f2$, $f4$, $f8$, and $f10$, the results of WOA and PWOAs maintain a tiny gap with optimal values (0). The proposed algorithms, particularly, can converge to a theoretical optimal value ($f_{\min} = 0$) for $f7$ and $f9$ at any scale. CSO can get more reliable solutions for $f1$, $f2$, $f8$, and $f9$, which, however, are still lower than the proposed algorithms in accuracy. Relatively speaking, PSO hardly converges to an accurate value for most benchmarks. The comparison between WOA and PWOAs shows that the proposed parallel framework based on FPGA heterogeneous platform for WOA maintains intrinsic outstanding global convergence. On top of that, with the increasing of both dimension and population size, the performance of the proposed algorithms are improved for most benchmark functions except for $f3$, $f5$, and $f8$, which indicates that dimension setting affects optimization performance to some

Table 1: Benchmark functions.

Func.	Expression	Range	$f_{\min}$
$f1$	$f(x) = \sum_{i=1}^{D} x_i^2$	$[-100, 100]^D$	0
$f2$	$f(x) = \sum_{i=1}^{D} \lvert x_i \rvert + \prod_{i=1}^{D} \lvert x_i \rvert$	$[-10, 10]^D$	0
$f3$	$f(x) = \sum_{i=1}^{D} (\sum_{j=1}^{i} x_j)^2$	$[-100, 100]^D$	0
$f4$	$f(x) = \max_{1 \le i \le D} \lvert x_i \rvert$	$[-100, 100]^D$	0
$f5$	$f(x) = \sum_{i=1}^{D-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$	$[-30, 30]^D$	0
$f6$	$f(x) = \sum_{i=1}^{D} -x_i \sin(\sqrt{\lvert x_i \rvert})$	$[-500, 500]^D$	$-418.983 \times D$
$f7$	$f(x) = \sum_{i=1}^{D} [x_i^2 - 10\cos(2\pi x_i) + 10]$	$[-5.12, 5.12]^D$	0
$f8$	$f(x) = -20e^{-0.02\sqrt{D^{-1}\sum_{i=1}^{D} - e^{D^{-1}\sum_{i=1}^{D}\cos(2\pi x_i)}}} + 20 + e$	$[-32, 32]^D$	0
$f9$	$f(x) = \sum_{i=1}^{D} (x_i^2/4000) - \prod_{i=1}^{D} \cos(x_i/\sqrt{i}) + 1$	$[-600, 600]^D$	0
$f10$	$f(x) = (\pi/D)\{10\sin(\pi y_1) + \sum_{i=1}^{D-1}(y_i - 1)^2[1 + 10\sin^2(\pi y_{i+1})] + (y_D - 1)^2\} + \sum_{i=1}^{D} u(x_i, 10, 100, 4)$, $y_i = 1 + (x_i + 1/4)$, $u(x_i, a, k, m) = \begin{cases} k(x_i - a)^m, & x_i > a, \\ 0, & -a < x_i < a, \\ k(-x_i - a)^m, & x_i < -a. \end{cases}$	$[-50, 50]^D$	0

Table 2: Comparison between the proposed algorithms and the state-of-the-art algorithms (64D).

Func.	PWOA-PPM		PWOA-AFPM		WOA		CSO		PSO	
	Mean	Time (s)	Mean	Time (s)	Mean	Time (s)	Mean	Time (s)	Mean	Time (s)
$f1$	**2.32E − 105**	0.1625	2.73E − 104	**0.1009**	1.98E − 103	0.2905	9.28E − 07	0.1715	7.52E − 07	0.2689
$f2$	**3.71E − 62**	0.1621	2.69E − 61	**0.0868**	6.50E − 61	0.3027	5.57E − 04	0.1746	8.98E + 01	0.2741
$f3$	6.36E + 02	0.1612	**3.47E + 02**	**0.1028**	2.67E + 03	0.2859	8.75E + 03	0.1711	1.76E + 04	0.2513
$f4$	9.16E − 16	0.1589	1.04E − 16	**0.9639**	**6.91E − 17**	0.2903	1.50E + 01	0.1598	3.42E + 01	0.2842
$f5$	2.73E − 01	0.1747	2.53E − 01	**0.1054**	**1.72E − 01**	0.2918	1.70E + 02	0.1803	3.22E + 02	0.2892
$f6$	**−2.65E + 04**	0.2068	−2.63E + 04	**0.1223**	−4.14E + 03	0.6707	−2.19E + 04	0.2945	−1.78E + 04	0.4320
$f7$	**0**	0.2123	**0**	**0.1139**	**0**	0.5681	6.57E + 01	0.2711	1.84E + 02	0.3750
$f8$	**2.04E − 15**	0.1978	2.40E − 15	**0.1098**	3.41E − 15	0.5659	1.60E − 04	0.2610	3.23E + 00	0.3799
$f9$	**0**	0.2016	**0**	**0.1076**	**0**	0.6759	2.30E − 03	0.2927	2.11E − 02	0.4254
$f10$	4.39E − 10	0.1685	3.93E − 10	**0.0945**	**1.35E − 10**	0.8494	6.37E − 02	0.3409	5.52E + 00	0.6659

Table 3: Comparison between the proposed algorithms and the state-of-the-art algorithms (128D).

Func.	PWOA-PPM		PWOA-AFPM		WOA		CSO		PSO	
	Mean	Time (s)	Mean	Time (s)	Mean	Time (s)	Mean	Time (s)	Mean	Time (s)
$f1$	**8.37E − 111**	**0.2614**	2.52E − 110	0.2781	1.21E − 110	1.1735	4.79E − 07	0.6871	1.08E − 01	1.0877
$f2$	**3.26E − 64**	**0.2588**	4.51E − 63	0.2761	1.71E − 62	1.1620	4.80E − 04	0.6495	4.05E + 02	1.1288
$f3$	2.40E + 03	**0.2584**	**1.68E + 03**	0.2817	2.45E − 03	1.1284	4.30E + 04	0.6805	9.83E + 04	1.3444
$f4$	**1.46E − 18**	**0.2540**	2.67E − 17	0.2693	1.70E − 17	1.1045	3.79E + 01	0.6723	5.37E + 01	1.1271
$f5$	4.36E − 01	0.2760	9.40E − 01	**0.2584**	**7.56E − 02**	1.1548	4.51E + 02	0.7085	7.63E + 02	1.1259
$f6$	−5.33E + 04	0.3525	**−5.35E + 04**	**0.3318**	−6.37E + 03	2.6482	−4.02E + 04	1.1221	V3.23E + 04	1.9468
$f7$	**0**	0.3321	**0**	**0.2641**	**0**	2.21	1.26E + 02	1.0309	4.43E + 02	1.7101
$f8$	4.58E − 15	0.3020	**2.22E-15**	**0.2798**	2.81E − 15	2.2070	8.17E − 04	1.0271	7.93E + 00	1.6865
$f9$	**0**	0.3193	**0**	**0.2733**	**0**	2.6825	1.11E − 02	1.0924	2.85E − 01	1.8733
$f10$	3.37E − 10	**0.2695**	2.69E − 11	0.3795	**2.01E − 11**	3.3434	4.35E − 01	1.3632	2.31E + 01	2.5813

extent. Generally speaking, benchmarking results of WOA, PWOAs, CSO, and PSO prove the effectiveness of two parallel WOAs proposed in this work.

Concerning running time, two perspectives of function type and scale settings are considered. From the perspective of function type, since multimodal functions ($f6 \sim f10$) generally have higher arithmetic complexity than unimodal functions ($f1 \sim f5$) [27, 40], there is an obvious time gap existed between unimodal and multimodal functions optimized by all algorithms in tables. For classic algorithms, WOA and PSO have relatively close running time, especially for unimodal functions. This is because the two algorithms essentially have similar structure and complexity. CSO, on the contrary, maintains faster performance than WOA and

Table 4: Comparison between the proposed algorithms and the state-of-the-art algorithms (256D).

Func.	PWOA-PPM		PWOA-AFPM		WOA		CSO		PSO	
	Mean	Time (s)	Mean	Time (s)	Mean	Time (s)	Mean	Time (s)	Mean	Time (s)
$f1$	**3.47E − 118**	**0.6760**	5.40E − 118	1.7067	4.72E − 118	4.4267	2.01E − 06	2.6118	7.82E + 02	4.3250
$f2$	**3.54E − 65**	**0.7065**	7.52E − 65	1.6059	5.59E − 65	4.5381	8.65E − 04	2.7718	8.64E + 02	4.5422
$f3$	**8.89E + 02**	**0.6839**	3.15E + 03	1.6452	2.87E + 03	4.4668	1.62E + 05	2.8792	4.13E + 05	4.5937
$f4$	7.75E − 21	**0.6947**	**2.20E − 21**	1.7199	1.04E − 20	4.3673	3.46E + 01	2.6202	6.84E + 01	4.5842
$f5$	3.81E + 00	**0.7995**	2.40E − 01	1.6395	**8.51E − 02**	4.5709	7.68E + 02	2.6535	2.0E + 05	4.9438
$f6$	−1.05E + 05	**1.0876**	**−1.07E + 05**	1.5272	−9.01E + 03	10.5611	−7.45E + 04	4.6981	−5.63E + 04	7.1954
$f7$	**0**	**1.1013**	**0**	1.6448	**0**	8.7863	1.93E + 02	4.1165	9.30E + 02	6.0118
$f8$	**1.06E − 15**	**0.9937**	2.98E − 15	1.6502	3.06E − 15	8.7506	8.34E − 04	4.0297	1.36E + 01	6.0041
$f9$	**0**	**1.0277**	**0**	1.6496	**0**	10.6604	1.31E − 03	4.3609	1.14E + 01	8.1001
$f10$	2.34E − 11	**0.8478**	**3.04E − 12**	1.6143	6.36E − 12	13.3073	5.20E − 01	5.2544	8.13E + 01	9.5807

Table 5: Comparison between the proposed algorithms and the state-of-the-art algorithms (512D).

Func.	PWOA-PPM		PWOA-AFPM		WOA		CSO		PSO	
	Mean	Time (s)	Mean	Time (s)	Mean	Time (s)	Mean	Time (s)	Mean	Time (s)
$f1$	**6.50E − 123**	**2.5753**	9.92E − 123	4.5841	1.38E − 122	17.6789	1.14E − 04	10.2368	8.32E + 04	18.1490
$f2$	**1.91E − 68**	**2.5890**	4.37E − 68	4.8587	9.88E − 68	18.0246	1.34E − 01	10.8658	1.74E + 03	19.0045
$f3$	1.73E + 04	**2.4585**	5.70E + 03	4.6452	**2.65E + 03**	17.8125	1.38E + 08	10.4507	1.54E + 06	18.5338
$f4$	**8.08E − 23**	**2.5920**	1.08E − 22	4.8721	4.79E − 22	17.4709	3.86E + 01	10.0652	9.81E + 01	18.7573
$f5$	**1.31E + 00**	**2.8612**	5.06E + 00	3.8336	7.35E + 00	18.2380	1.11E + 03	10.8096	1.00E + 08	18.8939
$f6$	−2.14E + 05	**3.1862**	**−2.17E + 05**	5.9001	−1.43E + 04	42.3103	−1.17E + 05	19.2030	−9.31E + 04	30.1889
$f7$	**0**	**3.0358**	**0**	3.9192	**0**	35.2929	2.79E + 03	17.0799	2.81E + 03	29.1872
$f8$	**6.13E − 16**	**2.9920**	3.29E − 15	5.6349	2.70E − 15	34.9978	4.36E − 03	15.6751	1.72E + 01	28.9203
$f9$	**0**	**3.0299**	**0**	4.3163	**0**	42.7160	5.22E − 04	18.2628	7.59E + 02	29.9871
$f10$	3.53E − 12	**2.9357**	**1.09E − 12**	7.4716	5.49E − 12	51.1879	3.34E − 01	21.4533	9.64E + 06	34.8555

PSO, where the simple and low-complexity algorithmic structures of CSO play an important part. In terms of PWOA-PPM and PWOA-AFPM, different scales of the problem have little impact on the execution time of these two algorithms. Regarding the problem scale, the running time of all three algorithms is affected by benchmark dimension and population size without exception. Canonical WOA is sensitive to problem scale, and different scale settings will cause a large gap in running time. For PWOA-PPM, the difference in execution time for functions with 64 D and 128 D is minimal but shows a trend of rapid growth for the scale of 256 D and 512 D. On the contrary, the performance of PWOA-PPM is relatively stable. As the scale increases, it demonstrates a slow growth of running time for PWOA-PPM.

In brief, the proposed PWOA-PPM and PWOA-AFPM are executed more stable than WOA, which benefits from the hardware-accelerated performance of FPGA due to built-in dedicated arithmetic units and modular design of the pipeline.

5.3. Speedup Analysis. In this section, speedup is calculated based on the running time of different problem scales, given as follows:

$$\text{Speedup} = \frac{T_{\text{WOA}}}{T_{\text{PWOA}}}, \tag{10}$$

where T_{WOA} and T_{PWOA} denote the running time of serial WOA and FPGA implementation of parallel WOA, respectively.

Speedup produced by PWOAs for settling various benchmark functions is shown in Figure 7 and analyzed as follows. Note that PWOAs have a certain degree of execution improvement and the speedup in both PWOA-PPM and PWOA-AFPM in multimodal functions is better than that in unimodal functions with all problem scales. From Figure 7(a), for both unimodal and multimodal functions, the greater the dimension of search space becomes, the higher the speedup ratio WOA-PPM obtains. Moreover, WOA-PPM can hold noticeable acceleration when solving the most complex $f10$, and the maximum speedup reaches around 18x with dimension = 512 D. As for WOA-AFPM, it has been found in Figure 7(b) that WOA-AFPM exhibits unstable computational performance, where the speedup for all functions decreases in case of 256 D while it manifests relatively better acceleration in the cases of 128 D and 512 D. In addition, the speedup ratio obtained by WOA-AFPM for optimizing $f10$, contrary to WOA-PPM, shows a slight downward trend, as the problem scale increases. The maximum speedup produced by PWOA-AFPM can be up to 10x (solving $f9$ with dimension = 512 D).

Four bar graphs, depicted in Figure 8, are used here to intuitively make comparisons for the speedup between PWOA-PPM and PWOA-AFPM with different problem scales. In cases of small scale including 64 D and 128 D, the speedup of PWOA-PPM is not as good as PWOA-AFPM, especially when solving all functions in case of 64 D and $f5 \sim f9$ in case of 128 D. Note, however, that the running efficiency of PWOA-PPM steadily rises as the scale

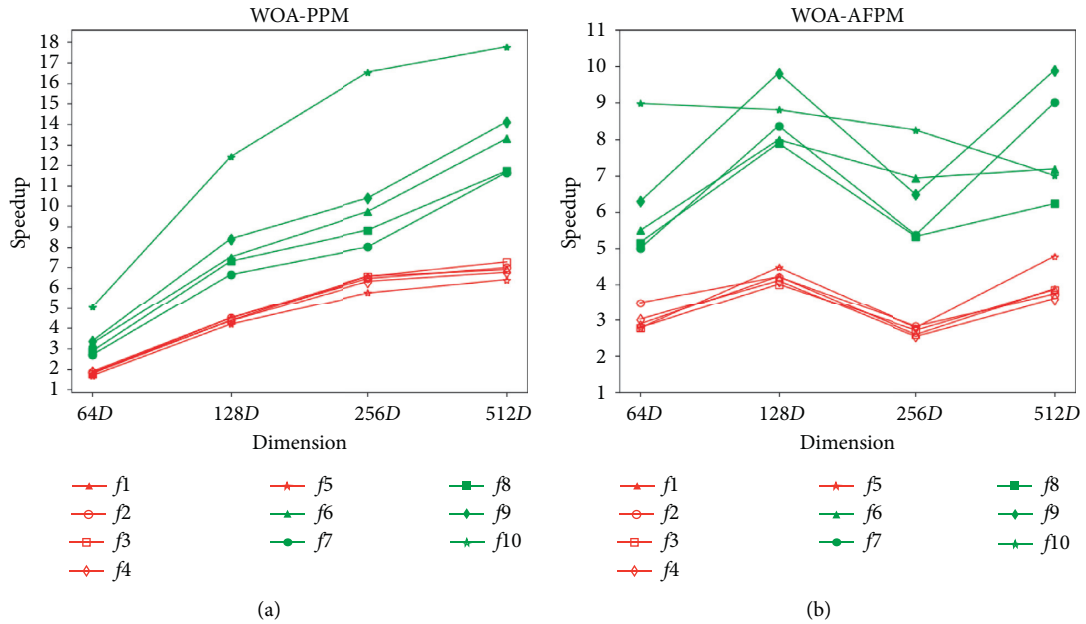

Figure 7: Speedup of PWOA-PPM and PWOA-AFPM w.r.t benchmark functions.

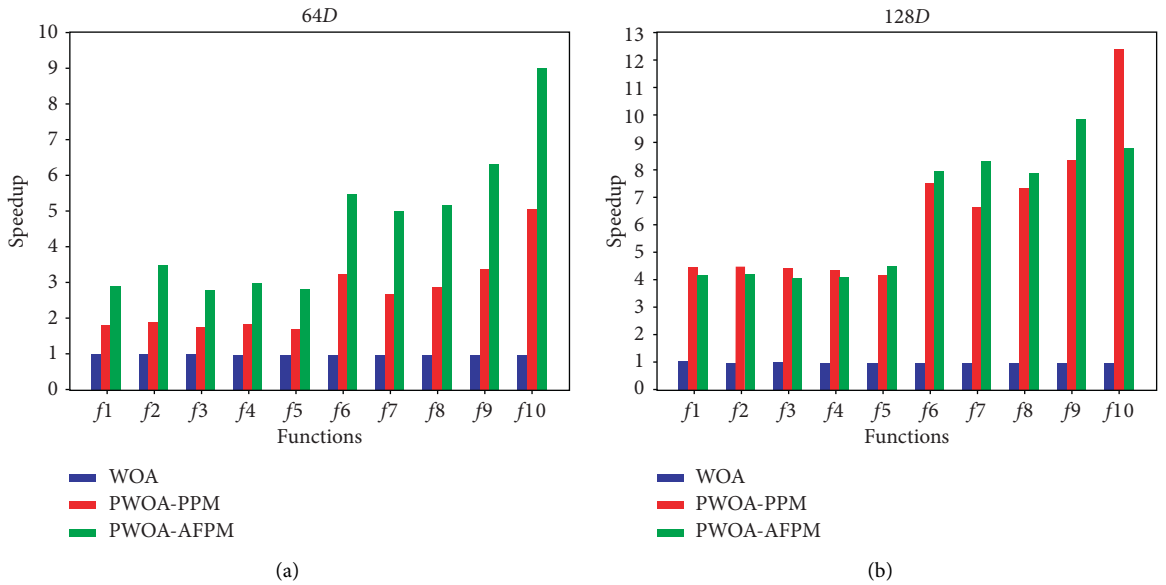

Figure 8: Continued.

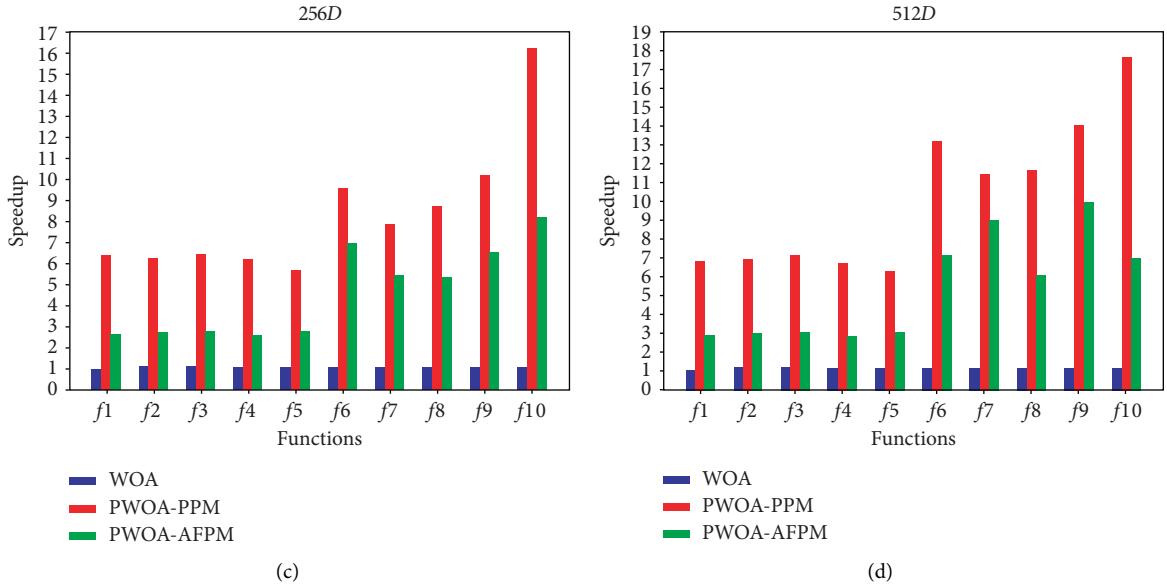

Figure 8: Comparison for the speedup between PWOA-PPM and PWOA-AFPM.

increases. In case of 256 D, the speedup of WOA-PPM for all unimodal functions and $f10$ is twice greater or more than that of WOA-AFPM. This value of speedup gap between WOA-PPM and WOA-AFPM further increases from twice to 2.5 times when solving $f10$ in case of 512 D. In a few words, WOA-PPM has more advantages in solving medium-scale and large-scale problems, while WOA-AFPM has better computational performance in small-scale problems.

It can be seen from above experimental analysis that PWOAs with two models have held discrepant influence on acceleration, which are mainly caused by the different frameworks instructing the implementation of PWOA-PPM and PWOA-AFPM on FPGA heterogeneous platform. For PWOA-PPM, it utilizes a partial parallel model and a certain amount of extra overhead that frequent communication between CPU and FPGA becomes a bottleneck leading to worse performance in case of small-scale. Unlike PWOA-PPM, PWOA-AFPM transfers most work of WOA to FPGA side for execution. Additionally, synchronous operation using memory fence requires more hardware to implement and might degrades kernel performance at FPGA side [55]. This, in turn, makes PWOA-AFPM become more inefficient with the increment of benchmark complexity and problem scale.

6. Conclusion

Demonstrating its excellence in global optimization, WOA has drawn significant research interests in the last few years. An unavoidable reality is that performance degradation takes places in WOA when facing large-scale complex optimization problems. Many proposals exist to address this issue, most of which, however, are based on classic algorithms such as genetic algorithm and particle swarm optimization, while very few literature studies about parallel WOA can be found. Based on FPGA accelerator, this study proposes two well-designed parallel models to implement parallel PWOA using the OpenCL framework with a demonstration on Intel heterogeneous platform. Finally, the performances of two parallel models based on PWOA (PWOA-PPM and PWOA-AFPM) have been evaluated using 10 benchmark functions.

For future work, it is essential to apply this algorithm to real engineering problems to verify the practical benefits. Besides, more different types of devices such as GPU and DSP need to be investigated, to build a multidevice heterogeneous platform. This platform will be an efficient cooperative running environment where a high-computational task can be decomposed into several parts and then assigned to different devices. Therefore, the proposed parallel scheme has potential for real applications.

Data Availability

The data used to support the findings of this study are included within the article.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

Acknowledgments

This research work was supported by the National Key Research and Development Project under grant no. 2018YFB1700500, National Science Foundation of China under grant nos. 52077213, 62003332, and 61702493, Natural Science Foundation of Guangdong (no. 2018A030310671), and Outstanding Young Researcher Innovation Fund of Shenzhen Institute of Advanced Technology, Chinese Academy of Sciences (no. 201822).

References

[1] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization," *Swarm Intelligence*, vol. 1, no. 1, pp. 33–57, 2007.

[2] M. Dorigo, M. Birattari, and T. Stutzle, "Ant colony optimization," *IEEE Computational Intelligence Magazine*, vol. 1, no. 4, pp. 28–39, 2006.

[3] S. Gao, Y. Yu, Y. Wang, J. Wang, J. Cheng, and M. Zhou, "Chaotic local search-based differential evolution algorithms for optimization," *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, vol. 99, p. 1, 2020.

[4] J. Sun, S. Gao, H. Dai, J. Cheng, M. Zhou, and J. Wang, "Bi-objective elite differential evolution algorithm for multivalued logic networks," *IEEE Transactions on Cybernetics*, vol. 50, no. 1, pp. 233–246, 2020.

[5] Y. Wang, Y. Yu, S. Gao, H. Pan, and G. Yang, "A hierarchical gravitational search algorithm with an effective gravitational constant," *Swarm and Evolutionary Computation*, vol. 46, pp. 118–139, 2019.

[6] S. Gao, M. Zhou, Y. Wang, J. Cheng, H. Yachi, and J. Wang, "Dendritic neuron model with effective learning algorithms for classification, approximation, and prediction," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 2, pp. 601–614, 2019.

[7] R. Cheng, M. Li, K. Li, and X. Yao, "Evolutionary multiobjective optimization-based multimodal optimization: fitness landscape approximation and peak detection," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 5, pp. 692–706, 2018.

[8] H. Cheng, S. Huang, R. Cheng, K. C. Tan, and Y. Jin, "Evolutionary multiobjective optimization driven by generative adversarial networks (GANs)," *IEEE Transactions on Cybernetics*, 2020.

[9] Y. Wang, Y. Yu, S. Cao, X. Zhang, and S. Gao, "A review of applications of artificial intelligent algorithms in wind farms," *Artificial Intelligence Review*, vol. 53, no. 5, pp. 3447–3500, 2020.

[10] S. Mirjalili and A. Lewis, "The whale optimization algorithm," *Advances in Engineering Software*, vol. 95, pp. 51–67, 2016.

[11] M. Abdel-Basset, G. Manogaran, D. El-Shahat, and S. Mirjalili, "A hybrid whale optimization algorithm based on local search strategy for the permutation flow shop scheduling problem," *Future Generation Computer Systems*, vol. 85, pp. 129–145, 2018.

[12] M. M. Mafarja and S. Mirjalili, "Hybrid whale optimization algorithm with simulated annealing for feature selection," *Neurocomputing*, vol. 260, pp. 302–312, 2017.

[13] I. Aljarah, H. Faris, and S. Mirjalili, "Optimizing connection weights in neural networks using the whale optimization algorithm," *Soft Computing*, vol. 22, no. 1, pp. 1–15, 2018.

[14] J. Wang, P. Du, T. Niu, and W. Yang, "A novel hybrid system based on a new proposed algorithm-multi-objective whale optimization algorithm for wind speed forecasting," *Applied Energy*, vol. 208, pp. 344–360, 2017.

[15] M. A. E. Aziz, A. A. Ewees, and A. E. Hassanien, "Multi-objective whale optimization algorithm for content-based image retrieval," *Multimedia Tools and Applications*, vol. 77, no. 19, pp. 26135–26172, 2018.

[16] A. Got, A. Moussaoui, and D. Zouache, "A guided population archive whale optimization algorithm for solving multi-objective optimization problems," *Expert Systems with Applications*, vol. 141, p. 112972, 2020.

[17] M. A. E. Aziz, A. A. Ewees, and A. E. Hassanien, "Whale optimization algorithm and moth-flame optimization for multilevel thresholding image segmentation," *Expert Systems with Applications*, vol. 83, pp. 242–256, 2017.

[18] A. E. Hassanien, M. Abd Elfattah, S. Aboulenin, G. Schaefer, S. Y. Zhu, and I. Korovin, "Historic handwritten manuscript binarisation using whale optimisation," in *Proceedings of 2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pp. 003842–003846, IEEE, Budapest, Hungary, October 2016.

[19] A. Mostafa, A. E. Hassanien, M. Houseni, and H. Hefny, "Liver segmentation in MRI images based on whale optimization algorithm," *Multimedia Tools and Applications*, vol. 76, no. 23, pp. 24931–24954, 2017.

[20] S. Harikarthik, V. Palanisamy, and P. Ramanathan, "Optimal test suite selection in regression testing with testcase prioritization using modified ann and whale optimization algorithm," *Cluster Computing*, vol. 22, no. 5, pp. 11425–11434, 2019.

[21] S. Raj and B. Bhattacharyya, "Optimal placement of TCSC and SVC for reactive power planning using whale optimization algorithm," *Swarm and Evolutionary Computation*, vol. 40, pp. 131–143, 2018.

[22] H. M. Hasanien, "Performance improvement of photovoltaic power systems using an optimal control strategy based on whale optimization algorithm," *Electric Power Systems Research*, vol. 157, pp. 168–176, 2018.

[23] S. Rahnamayan and G. G. Wang, "Solving large scale optimization problems by opposition-based differential evolution (ode)," *WSEAS Transactions on Computers*, vol. 7, no. 10, pp. 1792–1804, 2008.

[24] A. Jain, M. Mishra, S. K. Peddoju, and N. Jain, "Energy efficient computing-green cloud computing," in *Proceedings of 2013 International Conference on Energy Efficient Technologies for Sustainability*, pp. 978–982, IEEE, Nagercoil, April 2013.

[25] P. Liu, S. Li, and Q. Ding, "An energy-efficient accelerator based on hybrid CPU-FPGA devices for password recovery," *IEEE Transactions on Computers*, vol. 68, no. 2, pp. 170–181, 2018.

[26] Y. Zhou and Y. Tan, "Gpu-based parallel particle swarm optimization," in *Proceedings of 2009 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1493–1500, IEEE, Trondheim, Norway, May 2009.

[27] M. Jin and H. Lu, "Parallel particle swarm optimization with genetic communication strategy and its implementation on GPU," in *Proceedings of 2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems*, pp. 99–104, IEEE, Hangzhou, China, October 2012.

[28] D. Narjess and B. Sadok, "A new hybrid GPU-PSO approach for solving max-csps," in *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion*, pp. 119-120, Denver, CO, USA, July 2016.

[29] M. P. Wachowiak, M. C. Timson, and D. J. DuVal, "Adaptive particle swarm optimization with heterogeneous multicore parallelism and GPU acceleration," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 10, pp. 2784–2793, 2017.

[30] J. Kumar, L. Singh, and S. Paul, "GPU based parallel cooperative particle swarm optimization using C-CUDA: a case study," in *Proceedings of 2013 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE)*, pp. 1–8, IEEE, Hyderabad, July 2013.

[31] J. Hajewski and S. Oliveira, "Two simple tricks for fast cache-aware parallel particle swarm optimization," in *Proceedings of 2019 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1374–1381, IEEE, Wellington, New Zealand, March 2019.

[32] B. A. Menezes, H. Kuchen, H. A. A. Neto, and F. B. de Lima Neto, "Parallelization strategies for GPU-based ant colony optimization solving the traveling salesman problem," in *Proceedings of 2019 IEEE Congress on Evolutionary Computation (CEC)*, pp. 3094–3101, IEEE, Wellington, New Zealand, March 2019.

[33] Y. Djenouri, D. Djenouri, A. Belhadi, P. Fournier-Viger, J. Chun-Wei Lin, and A. Bendjoudi, "Exploiting GPU parallelism in improving bees swarm optimization for mining big transactional databases," *Information Sciences*, vol. 496, pp. 326–342, 2019.

[34] C. Jin and A. K. Qin, "A GPU-based implementation of brain storm optimization," in *Proceedings of 2017 IEEE Congress on Evolutionary Computation (CEC)*, pp. 2698–2705, IEEE, San Sebastian, Spain, June 2017.

[35] L. Ma, T. Zhang, R. Wang, G. Yang, and Y. Zhang, "Pbar: parallelized brain storm optimization for association rule mining," in *Proceedings of 2019 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1148–1156, IEEE, Wellington, New Zealand, March 2019.

[36] B. Chen, B. Chen, H. Liu, and X. Zhang, "A fast parallel genetic algorithm for graph coloring problem based on CUDA," in *Proceedings of 2015 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, pp. 145–148, IEEE, Xi'an, China, September 2015.

[37] Y. Ma and L. S. Indrusiak, "Hardware-accelerated parallel genetic algorithm for fitness functions with variable execution times," in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pp. 829–836, Denver, CO, USA, July 2016.

[38] H. Rico-Garcia, J.-L. Sanchez-Romero, A. Jimeno-Morenilla, H. Migallon-Gomis, H. Mora-Mora, and R. V. Rao, "Comparison of high performance parallel implementations of TLBO and JAYA optimization methods on manycore GPU," *IEEE Access*, vol. 7, pp. 133822–133831, 2019.

[39] Y. Khalil, M. Alshayeji, and I. Ahmad, "Distributed whale optimization algorithm based on mapreduce," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 1, p. e4872, 2019.

[40] Y. Tan and K. Ding, "A survey on GPU-based implementation of swarm intelligence algorithms," *IEEE Transactions on Cybernetics*, vol. 46, no. 9, pp. 2028–2041, 2016.

[41] F. A. Escobar, X. Chang, and C. Valderrama, "Suitability analysis of FPGAs for heterogeneous platforms in HPC," *IEEE Transactions on Parallel and Distributed Systems*, vol. 27, no. 2, pp. 600–612, 2015.

[42] H. M. Waidyasooriya, Y. Takei, S. Tatsumi, and M. Hariyama, "OpenCL-based FPGA-platform for stencil computation and its optimization methodology," *IEEE Transactions on Parallel and Distributed Systems*, vol. 28, no. 5, pp. 1390–1402, 2016.

[43] D. Weller, F. Oboril, D. Lukarski, J. Becker, and M. Tahoori, "Energy efficient scientific computing on FPGAs using OpenCL," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 247–256, Monterey, CF, USA, February 2017.

[44] K. Shata, M. K. Elteir, and A. A. EL-Zoghabi, "Optimized implementation of OpenCL kernels on FPGAs," *Journal of Systems Architecture*, vol. 97, pp. 491–505, 2019.

[45] D. Li, L. Huang, K. Wang, W. Pang, Y. Zhou, and R. Zhang, "A general framework for accelerating swarm intelligence algorithms on FPGAs, GPUS and multi-core CPUS," *IEEE Access*, vol. 6, pp. 72327–72344, 2018.

[46] H. M. Waidyasooriya, M. Hariyama, M. J. Miyama, and M. Ohzeki, "OpenCL-based design of an FPGA accelerator for quantum annealing simulation," *The Journal of Supercomputing*, vol. 75, no. 8, pp. 5019–5039, 2019.

[47] Q. Jiang, Y. Guo, Z. Yang, and X. Zhou, "A parallel whale optimization algorithm and its implementation on FPGA," in *Prceedings of 2020 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–8, IEEE, Glassgow, UK, July 2020.

[48] *OpenCL Overview*, 2019 https://www.khronos.org/opencl/</u>.

[49] *Intel FPGA SDK for OpenCL*, https://www.intel.com/content/www/us/en/software/programmable/sdk-for-opencl/overview.html</u>, 2019.

[50] D. Chen and D. Singh, "Using OpenCL to evaluate the efficiency of CPUS, GPUS and FPGAs for information filtering," in *Proceedings of 22nd International Conference on Field Programmable Logic and Applications (FPL)*, pp. 5–12, IEEE, Oslo, Norway, August 2012.

[51] J. Zhang and J. Li, "Improving the performance of OpenCL-based FPGA accelerator for convolutional neural network," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 25–34, Monterey, MX, USA, February 2017.

[52] U. Aydonat, S. O'Connell, D. Capalija, A. C. Ling, and G. R. Chiu, "An openclTM deep learning accelerator on arria 10," in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 55–64, Monterey, MX, USA, February 2017.

[53] N. Suda, V. Chandra, G. Dasika et al., "Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 16–25, Monterey, CF, USA, February 2016.

[54] *Intel Fpga Sdk for Opencl Pro Edition: Programming Guide*, 2019 https://www.intel.cn/content/dam/www/programmable/us/en/pdfs/literature/hb/opencl-sdk/aocl_programming_guide.pdf</u>.

[55] *Intel Fpga Sdk for Opencl Pro Edition: Best Practices Guide*, 2019 https://www.intel.cn/content/dam/www/programmable/us/en/pdfs/literature/hb/opencl-sdk/aocl-best-practices-guide.pdf</u>.

[56] M. Jamil and X. S. Yang, "A literature survey of benchmark functions for global optimisation problems," *International Journal of Mathematical Modelling and Numerical Optimisation*, vol. 4, no. 2, pp. 150–194, 2013.

[57] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95-International Conference on Neural Networks*, pp. 1942–1948, IEEE, Perth, UK, November 1995.

[58] R. Cheng and Y. Jin, "A competitive swarm optimizer for large scale optimization," *IEEE Transactions on Cybernetics*, vol. 45, no. 2, pp. 191–204, 2014.