Microprocessors and Microsystems 72 (2020) 102949

Contents lists available at ScienceDirect

Microprocessors and Microsystems

journal homepage: www.elsevier.com/locate/micpro

An analytical framework for high-speed hardware particle swarm optimization

Issam Damaj[a,*], Mohamed Elshafei[b], Mohammed El-Abd[c], Mehmet Emin Aydin[d]

[a] Electrical and Computer Engineering Department, Beirut Arab University, Debbieh, Lebanon
[b] Department of Computer Science and Software Engineering, Concordia University, Montreal, Canada
[c] Electrical and Computer Engineering Department, American University of Kuwait, Salmiya, Kuwait
[d] Computer Science and Creative Technologies Department, University of the West of England, Bristol, United Kingdom

ARTICLE INFO

Article history:
Received 16 January 2019
Revised 30 October 2019
Accepted 2 December 2019
Available online 3 December 2019

Keywords:
Particle swarm optimization
Hardware
Software
Performance
Analysis
Gate arrays

ABSTRACT

Engineering optimization techniques are computationally intensive and can challenge implementations on tightly-constrained embedded systems. Particle Swarm Optimization (*PSO*) is a well-known bio-inspired algorithm that is adopted in various applications, such as, transportation, robotics, energy, etc. In this paper, a high-speed *PSO* hardware processor is developed with focus on outperforming similar state-of-the-art implementations. In addition, the investigation comprises the development of an analytical framework that captures wide characteristics of optimization algorithm implementations, in hardware and software, using key simple and combined heterogeneous indicators. The framework proposes a combined Optimization Fitness Indicator that can classify the performance of *PSO* implementations when targeting different evaluation functions. The two targeted processing systems are Field Programmable Gate Arrays for hardware implementations and a high-end multi-core computer for software implementations. The investigation confirms the successful development of a *PSO* processor with appealing performance characteristics that outperforms recently presented implementations. The proposed hardware implementation attains 23,300 improvement ratio of execution times with an elliptic evaluation function. In addition, a speedup of 1777 times is achieved with a Shifted Schwefels function. Indeed, the developed framework successfully classifies *PSO* implementations according to multiple and heterogeneous properties for a variety of benchmark functions.

1. Introduction

Optimization is an important concept in the engineering domain [1–4]. Indeed, all engineering applications involve some sort of optimization in order to realize the final product. Optimization involves reducing cost, power consumption, time delay or increases the yield, profit, quality of solution, etc. As engineering problems became more challenging; with properties such as large size, discontinuity, non-differentiability, non-linearity, multi-objectiveness, and mixed variable types; it becomes essential to develop new optimization paradigms that can reach acceptable solutions in less time.

Meta-heuristics is a substantial and considerably important optimization paradigm that can be used to tackle nowadays engineering problems. A major class of meta-heuristics is Population-based algorithms, which update a population of solutions over a number of iterations until some stopping condition is satisfied. Population-based algorithms are categorized based on the inspiration behind their population update scheme. These categories include Evolutionary Algorithms and Swarm Intelligence algorithms. Due to the heavy computational workload constituting of optimization with population-based algorithms, computational speed matters for solving many dynamic and real-time problems with Evolutionary Algorithms and Swarm Intelligence algorithms. For example, radio resource scheduling for recent generations of wireless communication networks requires a decision to be made within nano-seconds level, which is not viable with conventional computing infrastructures [5,6]. Further research has been conducted over the past two decades in order to speed up the execution of such algorithms, whether by manipulating parameters, developing multiple/distributed versions or implementing them on specific hardware.

In this paper, we investigate the implementation of one Swarm Intelligence algorithm, namely Particle Swarm Optimiza-

* Corresponding author.
E-mail addresses: i.damaj@bau.edu.lb (I. Damaj), m_lshafe@encs.concordia.ca (M. Elshafei), melabd@auk.edu.kw (M. El-Abd), Mehmet.Aydin@uwe.ac.uk (M.E. Aydin).

https://doi.org/10.1016/j.micpro.2019.102949
0141-9331/

tion (*PSO*) [7,8], on Field Programmable Gate Arrays (*FPGAs*) [9–12]. The investigation benefits from the advancements in *FPGA* capabilities and aims at developing high-speed hardware cores. Such a hardware implementation enables embedding optimization algorithms in application to assist, or completely replace, a central processing component. The investigation is driven by the need for appealing performance characteristics, and the reliable operation within real-time applications, that hardware implementations can provide. Furthermore, the investigation comprises the development of a statistical analysis framework that captures wide characteristics of optimization computations. The proposed framework statistically combines the mathematical properties of optimization algorithms and their evaluation function complexities, besides, their implementations in *HW* and *SW*. To that end, the developed *PSO* implementations are employed to validate the proposed framework and verify its effectiveness in application. Indeed, limited work in the literature is found to analyze the performance of *PSO* implementations based on heterogeneous properties as in the proposed analysis framework. The general contribution of the paper is summarized as follows; a detailed discussion on the motivation, contributions, and objectives is presented in Section 2:

- Develop efficient hardware cores for the *PSO* algorithm on *FPGAs*.
- Develop an analytical framework that evaluates the fitness of optimization implementations based on heterogeneous performance indicators.

The rest of the paper is organized so that Section 2 present the motivation, research questions, and the paper contribution. In Section 3, a survey of the literature is presented. In Section 4, the processor design and implementation are presented. Section 5 introduces the statistical analysis model. A thorough performance evaluation is presented in Section 6. Section 7 concludes the paper and sets the ground for future work.

2. Research objectives

Challenges to optimization algorithms, including *PSO*, comprise performance characteristics (time, speed, efficiency, and complexity), storage requirements, reliability and accuracy, and first and foremost the dealing with the intrinsic sequential behavior of the algorithm. As related to *PSO*, the following research opportunities are highlighted:

- Identify and investigate the performance aspects of *SW* implementations of *PSO* while targeting high-performance multi-core processors.
- Develop efficient hardware cores for *PSO* under *FPGAs*.
- Identify and investigate the performance aspects of the *HW* implementations.
- Identify a set of performance indicators that aids the evaluation of *HW* and *SW PSO* implementations with different benchmark evaluation functions.
- Develop combined performance indicators for *PSO* that capture the qualitative and quantitative characteristics, and enables the classification of different implementations based-on combined *HW, SW*, and mathematical properties.
- Identify the type of problems that can be efficiently optimized using *PSO* based on heterogeneous *HW, SW*, and algorithmic properties.

The proposed investigation has several research objectives. The investigation aims at developing a high-speed hardware core for *PSO* on *FPGAs*. To this end, the focus is on outperforming similar state-of-the-art implementations reported in the literature under the same implementation environment, algorithm specifications, and the targeted set of Benchmark Evaluation Functions (*BEFs*). The hardware development includes the identification of effective implementation specifics and best practices. In addition, the investigation comprises the development of a statistical analysis model that captures wide characteristics of *PSO* implementations in *HW* and *SW*. Accordingly, the investigation includes the development of high-speed *SW* implementations on high-end multi-core processors. The analysis model comprises simple and combined performance indicators including a main indicator called the Optimization Fitness Indicator (*OFI*). The developed indicators analyze *PSO* implementations in terms of performance, hardware size, throughput, success rate, and combined forms. The aim of the *OFI* is to classify the performance of *PSO* implementations when targeting different *BEFs*. Therefore, the type of problems that can be best solved using *PSO* can be identified. The evaluation confirms the successful implementation of an *FPGA* core for the *PSO* with accelerated processing throughput for different *BEFs*. The developed combined indicators successfully classified implementations using a variety of benchmark functions according to desired properties.

In relation to the similar work presented in Section 3, the proposed developments enable the following comparisons:

- Compare the performance of partitioned versus nonpartitioned *FPGA* implementations. The comparison is done between the proposed all-in-hardware *FPGA* core with the partitioned implementation in [13]. Given that the same implementation environment, algorithm specifications, and the targeted set of *BEFs* are adopted in our investigation.
- Compare the performance of the proposed sequential core with parallel hardware versions of the *PSO*. The comparison includes the parallel-*PSO* version in [14] at a swarm population size of 8 particles. Besides, presenting a wider analysis than that presented in [14] to include populations of 16 and 32 particles. In addition, the comparison includes the multi-swarm parallel hardware implementation in [15].
- Present classifications of *PSO* implementations based on heterogeneous performance characteristics. The classifications enable straightforward identifications of the type of problems that can be best solved using a *PSO* implementation. Such straightforward identifications are proposed to replace tedious, intuitive, and multifaceted evaluations based on several single indicators—as usually adopted in the literature [13–20,20–22].

3. Related work

The emergence of high-performance *FPGAs* enabled their use in computationally-intensive applications, such as optimization. Many recent investigations are identified in the literature to target implementing the *PSO* algorithm on *FPGAs*. In this section, state-of-the-art related work is identified and presented, while closely-related work is thoroughly addressed in Section 6.2. In presenting similar work, we focus on a variety of implementation-specific aspects, such as, target devices, the used metrics to analyze the attained performance, and the main achievement of each investigation. Table 1 accompanies the discussion and provides a summary of findings.

Several attempts to achieve high processing speeds are presented in the literature [14,15,19,20,23]. Calazan et al. [14] propose a *HW* implementation of a parallel version of the *PSO* to run on a *Xilinx Virtex-6* as a co-processor. The approach aims to improve the execution of *PSO* so that the performance can be optimized for solving benchmark problems. The results demonstrate a significant speedup achieved through solving benchmark functions. In addition, Tewlode et al. [15,20,23] develop a direct *HW* implementation of *PSO* to solve the problem of emission source localization within the context of environment monitoring in [23], and extended with two more numerical benchmark functions in [20] to achieve signif-

Table 1
Summary of related works.

Ref.	Year	Implementation	Hardware	Metrics	Main achievement
[26]	2018	*PSO*; HW and SW	ARM and Xilinx FPGA ZedBoard	Hardware speedup; area utilization; execution time; period	Improving multiple indicators including hardware speedup per HW/SW implementation
[16]	2017	*PSO* and Genetic Algorithms	Xilinx Spartran-3	Area	Small-sized hardware area
[17]	2016	Neuro-Fuzzy system with *PSO*	Xilinx Virtex-5	Precision; accuracy of prediction	Improved accuracy
[19]	2014	*PSO*	Xilinx Virtex-5	Execution time	High processing speed
[14]	2014	Parallel *PSO*	Xilinx Virtex-6	Execution time; speedup; area	High processing speed
[15]	2012	Parallel *PSO*	Xilinx Spartran-3	No. of iterations; execution time	High processing speed
[18]	2011	*Re-exited PSO*	Altera Cyclone	Area; delay; power consumption; communication time between HW and SW	Optimizing performance per HW/SW partitions
[22]	2011	Core of *PSO* and other components	Altera Cyclone	Execution time; and no. of evaluations	Improved accuracy
[20]	2009	*PSO*	Xilinx Spartran-3	No. iterations; execution time; clock cycles	High processing speed
[23]	2008	*PSO*	Xilinx Spartran-3	No. Iterations, execution time; clock cycles	High processing speed
[24]	2007	*PSO*; HW/SW partitioning	N/A	A developed cost equation that captures result quality; delay; power	Optimizing result quality per HW/SW Partition

icant speedups. The speedup is studied with respect to execution time as well as the number of cycles, where the level of achievement is clearly indicated. The authors published further details of the study in [15], where parallel *HW* implementations under *Xilinx Spartan 3E FPGA* is compared to *MicroBlaze*-based *SW* implementations.

Improving accuracy using hardware implementations is the target of different investigations. Karakuzu et al. [17] present an implementation of neuro-fuzzy system trained with *PSO* to improve the learning and prediction performances. The implementation provides efficiency with avoiding some steps through the process in comparison to using look-up tables (*LUTs*). The proposed approach and implementation, using *Xilinx Virtex-5*, is tested with two scenario-based benchmark system identifications and a realistic number-plate recognition. The implemented neuro-fuzzy system model was trained with swarm intelligence algorithms including *ABC* and ordinary *PSO* alongside the proposed *PSO* approach. The proposed *HW* implementation achieves the expected level of solution quality using significantly less *HW* resources. Furthermore, Li et al. [22] introduce a *HW*/*SW* co-design approach for *PSO* based-on *SOPC* technique and pipeline design, where the algorithm is partitioned between *SW* and *HW* to possibly improve the performance when solving various numerical benchmark functions. The algorithm is partitioned so that the fitness evaluation is developed in *SW*, while particle updating is implemented in *HW* and referred to as Particle Updating Accelerator. The proposed implementation is created to run the *SW* part on a *Nios II CPU*, while the *HW* part to run on a *Cyclone II FPGA*. The results demonstrate that a speedup of 20 times is achieved over a *SW* implementation.

A variety of investigations, on performance evaluation of *PSO* implementations, is presented in the literature. Common investigations include comparing the performance of Genetic Algorithms and *PSO*, optimizing *HW*/*SW* partitioned implementations, and integrating hardware *PSO* in engineering applications. Ben Ameur and Sakly [16] present parallel *HW* implementations for *PSO* and Genetic Algorithms. The implementations are formulated with finite state machines and target a *Xilinx Spartan-3 FPGA*. The implementations are tested with a number of benchmark functions to compare *PSO* with Genetic Algorithms; the reported results are in favor of the *PSO* algorithm. In addition, Abdelhalim et al. [24] present constrained and unconstrained *HW*/*SW* partitioning problem with *SW*-based *PSO* implementations. Optimizing the intended system is done using various heuristic algorithms including Genetic Algorithms and *PSO*. The results and comparative analysis suggest that the proposed *PSO*, so-called *Re-exited PSO*, help achieving near-optimum results. The *Re-exited PSO* algorithm is used in [18] to solve partitioning problems for the *JPEG* encoding system introduced by Lee et al. [25]. The developed embedded system is implemented under a *Cyclone FPGA* as the co-processor of a main *Nios-II CPU*. The results are compared with findings of [25] to demonstrate the success of the proposed approach. The work presented in [26] proposes different approaches to implement *PSO* on *FPGAs*. To this end, the investigation targets an *FPGA* and an *ARM* processor in a *Xilinx Zynq-7000* system on chip. The analysis includes testing three benchmark functions. Moreover, Ettouil et al. present a *PSO* implementation that targets an extended set of evaluation functions [27]; the investigation includes thorough analyses of hardware area utilization and the attained clock period. Applications of hardware *PSO* include training neural networks [28], implementing Multiple-Input Multiple-Output (MIMO) detection systems [29], to name but a few.

4. Hardware design

An informal and systematic approach is adopted to develop hardware cores for the *PSO* algorithm [1,30]. The approach is informal in the sense that it does not rely on engineering formal methods [31]. In addition, the approach is systematic in the sense that its procedure can be reused to develop similar hardware solutions, however, the method is not yet automatic and does not include any code generations, compilations, or rapid prototyping of hardware circuits. Furthermore, the methodology is unified in the sense that it uses common software engineering techniques, such as flowcharts and state machines, to model the algorithm; accordingly, *HW* and *SW* designs are derived and implemented.

The development steps of the *HW* and *SW* implementations of the *PSO* are as follows:

1. Depict the algorithm using flowcharts.
2. Develop the software version.
3. Design the processor Datapath by identifying, allocating, and binding hardware resources at the register-transfer level [32].
4. Develop the Finite State Machine (*FSM*) of the control unit based on the flowchart.
5. Describe the developed hardware using a description language and synthesize the implementation for *FPGAs*.

The approach can be used to design partitioned hardware and software implementations of the *PSO* algorithm. However, the current use aims at developing independent hardware implementation for *FPGAs*. The adopted *PSO* is described in Algorithm 1 in pseudocode.

Algorithm 1 The adopted PSO algorithm.

Initialize particles
do
for i = 1 to Number of Particles **do**
  if $f(x_i) \leq f(p_{best_i})$ **then**
    $p_{best_i} = x_i$
    if $f(x_i) \leq f(gbest)$ **then**
      $gbest = x_i$
    end if
  end if
  for $j = 1$ to Number of Dimensions **do**
    $v_{ij}^{t+1} = wv_{ij}^{t} + c_1 r_1 (p_{best_{ij}}^{t} - x_{ij}^{t}) + c_2 r_2 (g_{best_j} - x_{ij}^{t})$ (1)
    if $v_{ij}^{t+1} > v_{\max}$ **then**
      $v_{ij}^{t+1} = v_{\max}$
    end if
    if $v_{ij}^{t+1} < v_{\min}$ **then**
      $v_{ij}^{t+1} = v_{\min}$
    end if
    $x_{ij}^{t+1} = x_{ij}^{t} + v_{ij}^{t+1}$ (2)
    if $x_{ij}^{t+1} > x_{\max}$ **then**
      $x_{ij}^{t+1} = x_{\max}$
      $v_{ij}^{t} = 0$
    end if
    if $x_{ij}^{t+1} < x_{\min}$ **then**
      $x_{ij}^{t+1} = x_{\min}$
      $v_{ij}^{t} = 0$
    end if
  end for
end for
While maximum iterations or minimum error is not reached

where,
c_1 and c_2 are the acceleration constants.
r_1 and r_2 are random numbers between 0 and 1.
$x_i(t)$ is the position of the ith particle at iteration t.
$v_i(t)$ is the velocity of the ith particle at iteration t.
w is the inertia weight factor.
g_{best} is the best position among all particles.
p_{best_i} is the i_{th} particle best position.
Equation 1 is the velocity update of the ith particle.
Equation 2 is for position update of the ith particle.

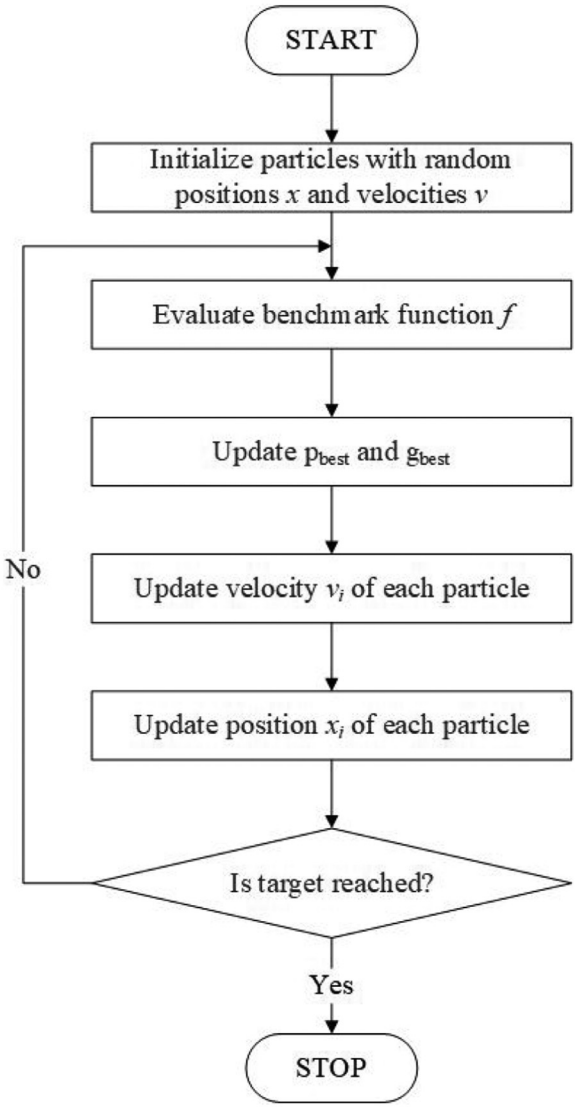


Fig. 1. An abstract flowchart for the *PSO* algorithm.

Fig. 1 depicts the abstract behavior of the *PSO* algorithm. The main actions in the algorithm are (i) generate random numbers, (ii) initialize particles positions and velocities, (iii) evaluate a benchmark function to calculate the fitness value of a particle, (iv) update the particle's record with the best fitness value attained yet, and (v) update the swarm's global record with the best fitness value reached yet. At this point, update the particle's velocity and position with newly calculated values plays the main role. The procedure is repeated until a target number of iterations or a minimum error value is reached.

Based on to the *PSO* algorithm description, the datapath development includes the allocation of several computational hardware resources (See Fig. 2). The main allocated hardware units are a Fitness unit, p_{best} Update unit, g_{best} Update unit, Velocity Update unit, Position Update unit, Random Number Generator (*RNG*), and three delay registers for the particle position x. Sample internal organizations of *PSO* computational units are shown in Figs. 3 and 4. In Fig. 3, the hardware structure is for the Velocity Update Unit, while Fig. 4 presents a Fitness Unit depicting the Rosenbrock function. In both figures, the computational components run in parallel. Moreover, Fig. 2 includes standard components, such as, registers and memory elements; the units Position, Global, and Particle Updates are implemented using simple selection statements, variable assignments, or include a single operation. In the presented datapath, the Random Number Generator core is imported from the work presented in [33].

The behavior of the processor control unit is described in the *FSM* shown in Fig. 5. States S_0 through S_6 are responsible for the position and velocity initialization of particles, preparation for the random number generation, and the forwarding of the position values through the three delay registers. State S_7 is the main execution procedure that includes benchmark function evaluation, best positions updates, and velocity and position updates. In addition, state S_7 repeats unit the stoppage criterion is satisfied.

5. Analytical model development

To present the proposed performance analysis model, we adopt the Generic Benchmark Model (*GBM*) of Damaj et al. [34]. The *GBM* comprises six elements that define the Goal, Inputs, Activities, Output, Outcomes, and the desired Performance profile of the performance analysis framework. The model captures the relationships

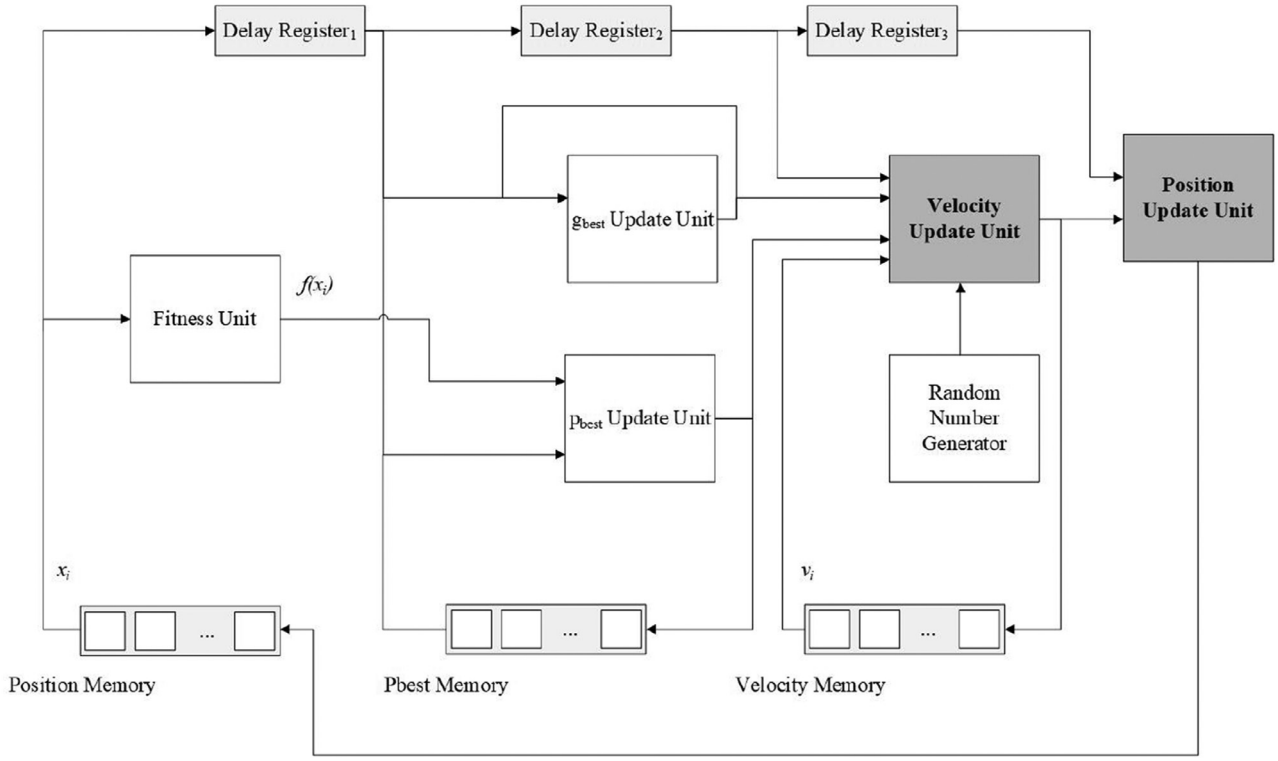


Fig. 2. The developed datapath for the *PSO* algorithm.

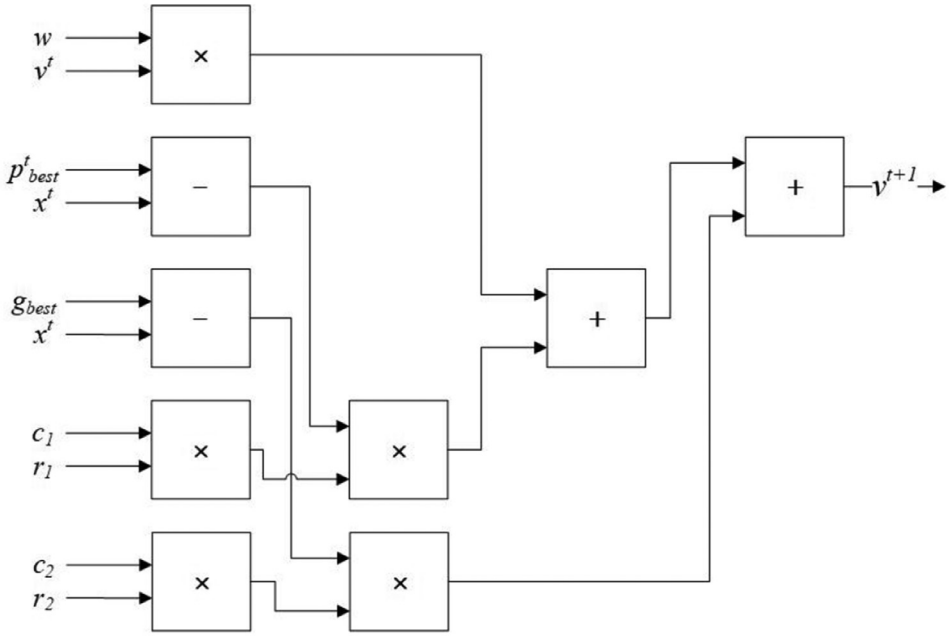


Fig. 3. The hardware structure of the Velocity Update Unit.

among the resources, implementation, mathematical formulation, and the obtained results. The Goal defines the aim of the analysis framework. Moreover, the Input identifies the algorithms under study, implementation environments, reference algorithm, performance metrics, etc. Furthermore, Activities present the algorithm implementations and the obtained results. The Output is the formulation of the key indicators and development of their rubrics—if needed. The Outcomes are the formulations of the statistical assessment as combinations of the Output. In addition, the Performance is the application of the developed assessment framework to profile and classify algorithms according to the obtained results.

5.1. Goal

Analyze the performance of a *PSO* implementations in *HW* and *SW*, and enable its comparison to similar work in the literature.

5.2. Input

The input identifies the targeted algorithms, computing systems, and the performance metrics. The analysis model targets a set of benchmark evaluation functions of different complexities and characteristics (See Tables 2 and 3). Moreover, the targeted

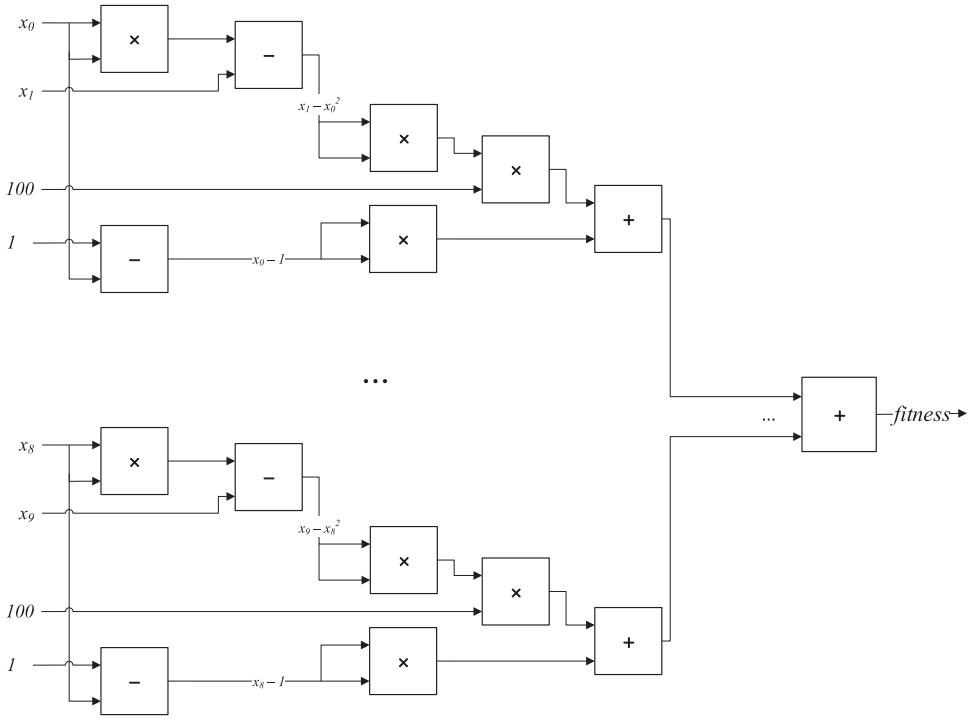


Fig. 4. The hardware structure of the Fitness Unit depicting a Rosenbrock function.

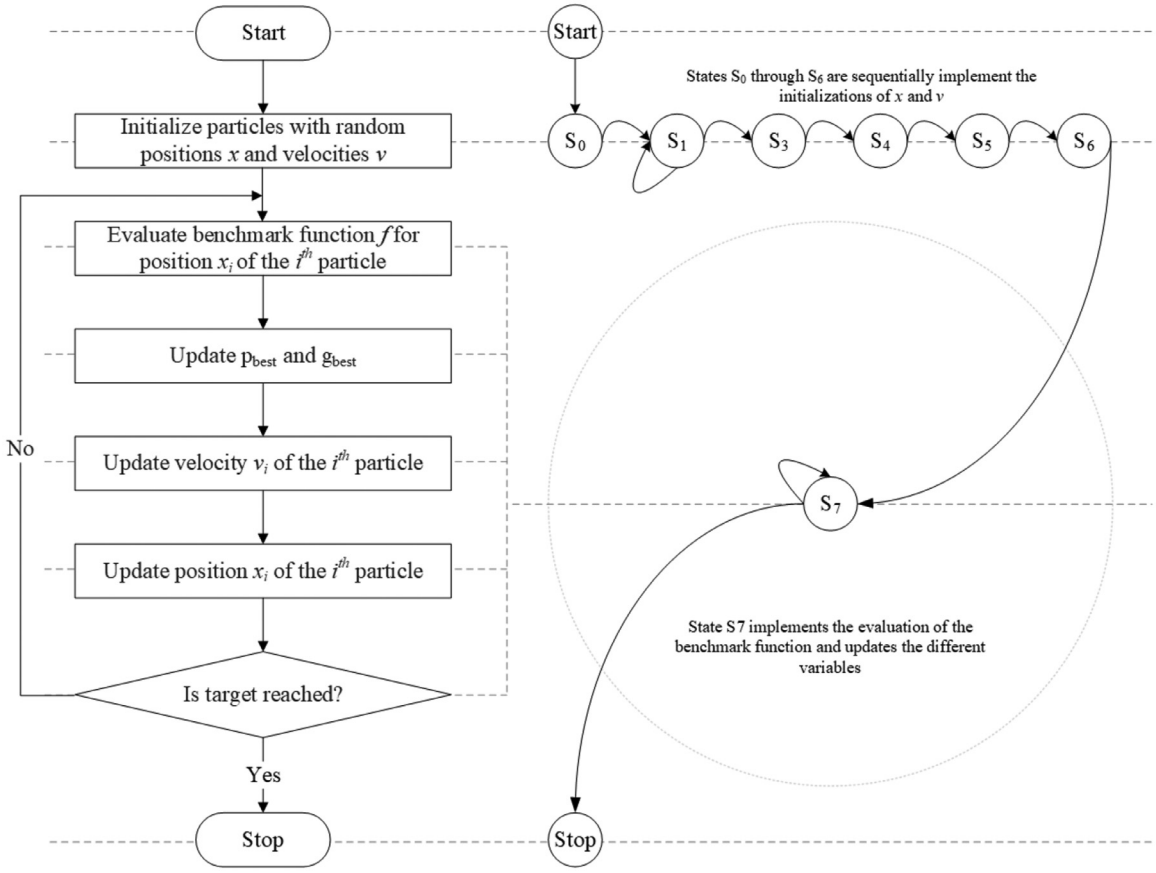


Fig. 5. The developed *FSM* of the *PSO* Algorithm and its correspondence to the flowchart steps.

Table 2
The targeted benchmark evaluation functions.

BEF Index	Function	Variable no.	Search domain	Optimal solution
F1	B2	2	[−100, 100]	0
F2	Branin	2	[−4, 4]	0.397887
F3	Goldstein−Price	2	[−2, 2]	3
F4	Rosenbrock	2	[−9, 11]	0
F5	Zakharov	2	[−10, 10]	0
F6	Sphere	3	[−5.12, 5.12]	0
F7	Hartmann	3	[0, 1]	−3.863433
F8	Variably−dimensioned	4	[−9, 11]	0
F9	Shifted Sphere	32	[−100, 100]	0
F10	Shifted Rosenbrock	32	[−100, 100]	0
F11	Shifted Schwefel 1.2	32	[−100, 100]	0
F12	Shifted Rastrigin	32	[−100, 100]	0
F13	Shifted Rotated High Conditioned Elliptic	32	[−100, 100]	0

Table 3
Properties of the target benchmark evaluation functions in terms of separability, scalability, and whether the function is unimodal or multimodal.

BEF Index	Function	Separable	Scalable	Unimodal
F1	B2	Yes	No	No
F2	Branin	No	No	No
F3	Goldstein-Price	No	No	No
F4	Rosenbrock	No	Yes	No
F5	Zakharov	Yes	Yes	Yes
F6	Sphere	Yes	Yes	Yes
F7	Hartmann	No	No	No
F8	Variably-dimensioned	Yes	Yes	Yes
F9	Shifted Sphere	Yes	Yes	Yes
F10	Shifted Rosenbrock	No	Yes	Yes
F11	Shifted Schwefel 1.2	No	Yes	Yes
F12	Shifted Rastrigin	Yes	Yes	No
F13	Shifted Rotated High Conditioned Elliptic	No	Yes	Yes

HW development board is the DE2-70 by *Altera*. The board has a *Cyclone II FPGA* with a total of 68,416 Logic Elements (LEs) and a maximum frequency of 300 MHz. *SW* implementations are done on *Dell Precision T7500* with its dual quad-core *Xeon* processors and *24 GB* of *RAM*.

The identified performance metrics of the *PSO* are classified into general algorithmic profile (*GAP*), hardware profile (*HWP*), and software profile (*SWP*). The general algorithmic profile includes the complexity of the benchmark evaluation functions. The *HWP* includes the number of benchmark evaluations, resource utilization, maximum frequency, throughput, etc. Moreover, the *SWP* comprises the number of benchmark evaluations, throughput, execution time, etc.

5.3. Activities

The activities include hardware implementations under *VHDL*. The Software tools used for hardware implementation and profiling are *Quartus* and *ModelSim*. Software implementations are done under *MATLAB*.

5.4. Output

The outputs of the analysis framework are three sets of indicators that correspond to the proposed *GAP, HWP*, and *SWP*. The main *KI* of the *GAP* is the Benchmark Evaluation Function Complexity (*BEFC*), which is defined as follows:

- **Benchmark Evaluation Function Complexity** (*BEFC*): an asymptotic complexity analysis using the Big-*O*, small-ω, and Θ notations.

To analyze the complexity of the evaluation functions, we study their asymptotic behavior. The asymptotic behavior classifies algorithms according to their rate of growth with respect to the increase in input size. The following standard complexity analysis classification is adopted from [34,35]:

- $O(f(n))$: The rate of growth of an algorithm is asymptotically no worse than the function $f(n)$ but can be equal to it.
- $\omega(f(n))$: The rate of growth of an algorithm is asymptotically no better than the function $f(n)$.
- $\Theta(f(n))$: The rate of growth of an algorithm is asymptotically equal to the function $f(n)$.

Here, n is the size of input.

To facilitate the assessment of the studied ciphers, we adopt the rubric from [34] as shown in Table 4. In preparation for the statistical formulation, we map this qualitative properties onto quantities. For every point in the scale, we map it onto a fixed number. Hence, each point in the scale is mapped onto the values 20%, 40%, 60%, 80%, and 100% [34].

The hardware profile includes the following indicators:

- **Number of Benchmark Function Evaluations** (*NBFE*): the total number of calls of the benchmark function; equals the number of iterations times the total number of benchmark function calls per iteration. In this investigation, we report the average *NBFE* required for optimizing the benchmark functions.
- **Execution Time** (*ET*): the time between the start and the completion of a task.
- **Throughput** (*TH*): the total amount of work done in a given time. In this investigation, we calculate *TH* as the *NBFE* divided by *ET*; the results are represented in *Kilo BFE* per *Seconds* (*KBFEps*).
- **Logic Elements** (*LE*): the number of combinational logic elements required to implement an algorithm in hardware. The number of *LEs* is an indicator of the size of hardware in Altera devices.

Table 4
The rubric of the complexity analysis indicator.

General	Scale				
Indicator	Logarithmic low	Logarithmic high	Linear	Almost quadratic	Higher than quadratic
Complexity analysis	$O(logn)$	$\omega(logn)$ but better than Linear	$\Theta(n)$	$O(n^2)$ but worse than Linear	$\omega(n^2)$

- **Logic Register** (*LR*): the total number of logic registers in the design.

The Software profile includes three indicators, namely, ***NBFE, ET***, and ***TH***.

5.5. Outcomes

The Outcomes element is the formulation of Combined Measurement Indicators *CMIs* as function of *KIs*. Four *CMIs* are developed to analyze the performance of the *PSO* implementation, namely, **Success Rate** (*SR*), **Performance Rate** (*PR*), **Success Rate Density** (*SRD*), and the **Optimization Fitness Indicator** (*OFI*). The definitions of the *SR, PR*, and *SRD* are as follows:

- **SR**: The percentage number of runs that successfully converges to the minimum which is below the specified error divided by the total number of runs.
- **PR**: The average *NBFE* divided by the percentage *SR*.
- **SRD**: The number of *LEs* divided by *SR*. *SRD* captures the size of hardware used per 1% *SR*.

The *OFI* is the main *CMI* calculation in the presented statistical analysis model. A higher *OFI* is achieved through a higher throughput, a lower execution time, with less resource utilization, and at a higher performance rate; while targeting the evaluation function with a higher complexity. The combination of indicators is done using the Geometric Mean of *KI* ratios. The generic equation of *CMIs* from [34] is as follows:

$$CMI = \sqrt[n]{ratio_1 \times ratio_2 \times \dots ratio_n}$$

$$\text{Where } ratio_i = \frac{KI_{i,j}}{KI_{i,j}^{ref}}$$

$KI_{i,j}$ is the ith *KI* of the jth *profile*,

$i \in \{1 \dots n\}$ and $j \in \{1 \dots 2\}$,

and $KI_{i,j}^{ref}$ is the reference measurement of the indicator $KI_{i,j}$

To calculate a *CMI*, the Geometric Mean is used as it is able to measure the central tendency of data values that are obtained from ratios. The attraction for using the Geometric Mean is that its ratio is equal to the Geometric Mean of performance ratios; which implies that when comparing two different implementations' performance, the choice of the reference implementation is irrelevant [34,36]. In the current investigation, the reference measurements are considered as an evaluation function that attains an average performance as compared to the targeted *BEFs*. In other words, the reference measurement is calculated as the arithmetic average of results achieved by the targeted *BEFs*.

The *OFI* enables the classification of *PSO* algorithm according to its fitness in application. The *OFI* is either directly or inversely proportional to the indicators. The master *OFI* formula, using the developed indicators, is shown in Eq. (1). The indicators that are common to the Software (*sw*) and Hardware (*hw*) profiles are labeled with the profile name.

$$OFI = \sqrt[9]{GAP \cdot HWP \cdot SWP} \tag{1}$$

where,

$$GAP = \frac{BEFC}{BEFC_{ref}} \tag{2}$$

Table 5
PSO execution parameters.

Parameter	Values
Particle Coding	Binary
No. of Bits of Variables	8
Population Size	8, 16, 32
No. of Independent Runs	100
Maximal Function Evaluations	10,000 for F1–F8 200,000 for F9–F13
$c_1.r_1$	0–2
$c_2.r_2$	0–2
Inertia Weight w	0.25
Termination Error Threshold	$< 10^{-4}$

$$HWP = \frac{ET_{hw,ref}}{ET_{hw}} \cdot \frac{TH_{hw}}{TH_{hw,ref}} \cdot \frac{LE_{ref}}{LE} \cdot \frac{LR_{ref}}{LR} \cdot \frac{SR_{hw}}{SR_{hw,ref}} \tag{3}$$

$$SWP = \frac{ET_{sw,ref}}{ET_{sw}} \cdot \frac{TH_{sw}}{TH_{sw,ref}} \cdot \frac{SR_{sw}}{SR_{sw,ref}} \tag{4}$$

Besides the main *OFI*, two additional *CMIs* are proposed in Eqs. (5) and (6) to separately capture the optimization fitness of *HW* and *SW*:

$$OFI_{hw} = \sqrt[6]{GAP \cdot HWP} \tag{5}$$

and,

$$OFI_{sw} = \sqrt[4]{GAP \cdot SWP} \tag{6}$$

5.6. Performance

The analysis based on the *OFI Output* and *Outcomes* provides measurements for all *KIs* and enables the calculation of the defined *CMIs*. The results enable classifying the targeted evaluations. The six elements of the *OFI* are summarized in Fig. 6.

6. Analysis and evaluation

The analysis of the developed *PSO* implementations are produced to serve for several evaluation purposes. Important implementation aspects are presented in Section 6.1. In addition, a set of thirteen *BEFs*, namely *F1* through *F13*, is targeted to perfectly match the test-cases of the work presented in [13] and enable comparisons with similar work. To that end, the *HW* analysis is presented in Section 6.2 with comparisons to similar findings from closely-related work in the literature. A second set of six *BEFs*; namely *F1, F3, F4, F5, F6*, and *F8*; is targeted to compare the *HW* and *SW* implementations and evaluate the effectiveness of the proposed *CMIs* (See Sections 6.3). The analysis is done using the developed *KIs* and *CMIs*. The values of the execution parameters are shown in Table 5. In addition, Section 6.4 presents a thorough general evaluation of achievements of the research objectives.

6.1. Implementation aspects

A variety of tools is used to develop, validate, and analyze the hardware implementations. The targeted *FPGA* is *Cyclone II* by *Altera*. The tools used for hardware syntehsis and analysis are *Quartus* and *ModelSim*. The hardware implementations are done un-

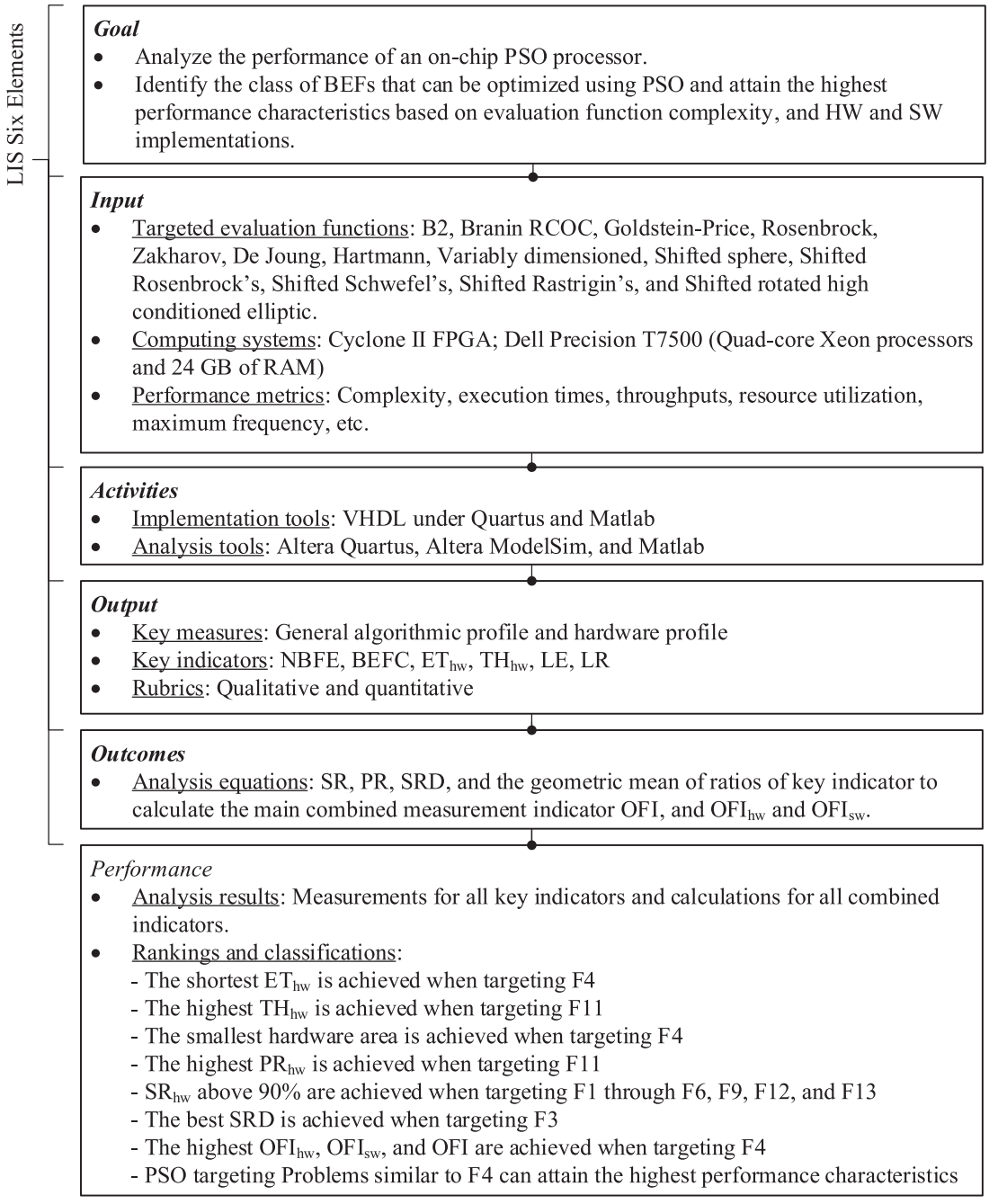


Fig. 6. The six elements diagram of the *PSO* performance analysis model.

der *VHDL*. The adopted *VHDL* style mixes structural and behavioral implementations. The units Fitness Module, p_{best} Update, g_{best} Update, Velocity Update, Position Update, *RNG*, and the three delay registers are implemented as separate *VHDL* components. The *RNG* is implemented using neighborhood-of-four cellular automata for *FPGAs* [33]. In the proposed hardware implementation, the fixed-point package from [37] is used to express floating-point numbers; the implementation varies in widths among the different computational entities. The *ieee_proposed.fixed_pkg.ALL* and *ieee_proposed.math_utility_pkg.ALL* VHDL libraries are employed, where the highest utilized width is of 64 bits and the highest adopted precision is with a fraction part of 9 bits. The adopted fixed-point representation provides a set of efficient operations that can replace an intrinsically complex floating-point alternative at a compromised but adequate accuracy per context.

6.2. Hardware performance analysis

6.2.1. Partitioned versus nonpartitioned hardware implementations

Li et al. [13] adopt a *HW/SW* co-design approach to improve the execution performance of *PSO* for embedded applications. The investigation targets the *DE2-70* board from *Altera* with its *Cyclone II FPGA* for *HW* implementations, and *Nios II* processor for *SW* implementations. The main features of the presented approach comprise partitioned *HW* and *SW* implementations of the *PSO*. The main proposed system components are a partitioned *HW* and *SW* evaluation, and a *HW* particle updating accelerator. The *HW* and *SW* subsystems communicate through the board interfaces and use an on-board *SDRAM*. Furthermore, the features include the design of a *HW RNG*. Experimental results demonstrate that the proposed *HW* design attains adequate efficiency and accuracy. The reported re-

Table 6
Hardware profile: *ET* and *TH; BEFs* F1 through F13 are included to enable the comparison with the work presented in [13].

	ET(msec) per population size			TH(KBFEps) per population size		
BEF	8	16	32	8	16	32
F1	14.04	28.84	59.27	8.83	5.89	4.57
F2	5.41	10.37	22.46	23.66	26.52	14.16
F3	8.29	16.14	34.77	23.88	17.29	8.86
F4	2.31	3.89	9.23	477.49	287.15	103.47
F5	20.87	42.64	88.35	6.76	3.94	3.62
F6	25.31	51.91	107.28	9.25	5.82	4.25
F7	29.63	60.93	125.69	8.07	5.58	4.18
F8	32.82	67.55	139.2	27.79	18.76	12.22
F9	57.85	119.81	245.9	909.96	452.55	227.11
F10	66.01	136.85	280.68	506.70	251.74	126.42
F11	62.12	128.73	264.12	1777.48	883.48	443.52
F12	66.99	138.89	284.86	523.20	259.92	130.53
F13	71.24	147.77	302.99	380.81	189.10	94.99

Table 7
Hardware profile: *LE* with percent hardware utilization and *LR*. The percent hardware utilization is calculated as the ratio of measured LEs divided by 68416, which is the total number of LEs available in the target FPGA. *BEFs* F1 through F13 are included to enable the comparison with the work presented in [13].

	LE per population size			LR per population size		
BEF	8	16	32	8	16	32
F1	$10340_{15.1\%}$	$11534_{16.9\%}$	$12770_{18.7\%}$	239	278	309
F2	$9753_{14.3\%}$	$10882_{15.9\%}$	$12048_{17.6\%}$	217	254	282
F3	$9948_{14.5\%}$	$11098_{16.2\%}$	$12288_{18\%}$	224	262	291
F4	$6055_{8.9\%}$	$6774_{9.9\%}$	$7500_{11.0\%}$	74	95	105
F5	$10943_{16.1\%}$	$12203_{17.8\%}$	$13512_{19.7\%}$	262	304	338
F6	$12486_{18.3\%}$	$13917_{20.3\%}$	$15409_{22.5\%}$	322	370	412
F7	$14171_{20.7\%}$	$15789_{23.1\%}$	$17482_{25.6\%}$	387	443	493
F8	$15756_{23.1\%}$	$17549_{25.7\%}$	$19432_{28.4\%}$	448	510	569
F9	$42295_{61.8\%}$	$47022_{68.7\%}$	$52075_{76.1\%}$	1472	1647	1838
F10	$45608_{66.7\%}$	$50702_{74.1\%}$	$56150_{81.1\%}$	1599	1788	1996
F11	$43964_{64.3\%}$	$48876_{71.4\%}$	$54127_{79.1\%}$	1536	1718	1918
F12	$47056_{68.8\%}$	$52310_{76.5\%}$	$57931_{84.7\%}$	1655	1850	2065
F13	$48725_{71.2\%}$	$54164_{79.2\%}$	$59984_{87.7\%}$	1719	1921	2145

Table 8
Hardware *CMIs:* SR_{hw} and PR_{hw}*; BEFs F1* through *F13* are included to enable the comparison with the work presented in [13].

	SR_{hw}% per population size			PR_{hw} per population size		
BEF	8	16	32	8	16	32
F1	84	98	96	1.48	1.73	2.82
F2	90	91	100	1.42	3.02	3.18
F3	77	89	95	2.57	3.13	3.24
F4	56	83	95	19.7	13.46	10.05
F5	92	95	89	1.53	1.77	3.60
F6	76	93	90	3.08	3.25	5.07
F7	45	77	87	5.31	4.42	6.05
F8	24	81	93	38	15.64	18.29
F9	100	100	100	526.41	542.20	558.47
F10	62	69	83	539.47	499.28	427.52
F11	31	45	59	3561.84	2527.33	1985.46
F12	100	100	100	350.49	361.00	371.83
F13	100	100	100	271.29	279.43	287.81

Table 9
Hardware *CMIs: SRD*; the *BEFs F1* through *F13* are included to enable the comparison with the work presented in [13].

	SRD per population size		
BEF	8	16	32
F1	123.10	117.69	133.02
F2	108.37	119.58	120.48
F3	129.19	124.70	129.35
F4	108.13	81.61	78.95
F5	118.95	128.45	151.82
F6	164.29	149.65	171.21
F7	314.91	205.05	200.94
F8	656.50	216.65	208.95
F9	422.95	470.22	520.75
F10	735.61	734.81	676.51
F11	1418.19	1086.13	917.41
F12	470.56	523.10	579.31
F13	487.25	541.64	599.84

sults, of the partitioned implementations, are compared with *SW* implementations.

In our investigation, exactly the same development board, *FPGA, BEFs*, and the execution parameters of [13] are targeted—to enable a sound comparison. However, we provide a fully-autonomous *HW* implementation on *FPGAs* and adopt a different *RNG* algorithm (See Section 4). Tables 6 and 7 present the results of the developed *HW* implementation. Here, thirteen *BEFs* are targeted to enable the comparison with the work reported in [13]. Table 6 shows that the shortest ET_{hw} of 2.31 ms is achieved when targeting *F4*, and the highest TH_{hw} of 1777.48 *KBFEps* is attained when targeting *F11*. The smallest attained hardware area is for targeting *F4* with 6055 *LEs* and 74 *LRs*. On the other hand, the longest ET_{hw} is taken by *F13* with a maximum value of 302.99 ms for a population size of 32. Moreover, the lowest TH_{hw} is attained by targeting *F5* with a population size of 32. The largest *HW* areas are required by *F13*. The highest PR_{hw} of 3561.84 is reached when targeting *F11*; here, the smallest is attained by *F2* with a value of 1.42. To that end, the function that reached the highest *SRD* is *F11* with a maximum of 1418.19 at a population size of 8.

In this investigation, adopting the same *FPGA* board and execution parameters as in [13] supports the conclusion that the achieved performance gains are due to the differences in the proposed *HW* system architectures. With no doubt, the communication cost between the partitioned *HW* and *SW* subsystems affects the overall performance as reported in [13]. In addition, the communication cost increases with the increase in *NBFE*; such an increase in communication further degrades the performance of the developed system. A replacement all-*HW* system is proposed by the authors that targets *F9*. The all-*HW* achieves an *ET* of 13.4673 ms with a performance improvement ratio of 12,115 over their partitioned implementation and 4.29 over our implementation.

In Tables 8 and 9, the SR_{hw}, PR_{hw}, and *SRD* are shown. Targeting the evaluation functions *F2, F9, F12*, and *F13* enables the achievement of the best *SR* value of 100%; while *F7, F8*, and *F11* achieve the smallest values with a minimum of 24% for *F8* with a population size of 8. Indeed, the results show that the best SR_{hw} and PR_{hw} are achieved, in most cases, at a population size of 32. Furthermore, the evaluation function *F4* achieves the lowest *SRD* score; this reflects the smallest hardware area utilization per percent success among all populations.

The results achieved by the proposed hardware implementation show significant improvements, in terms of the measured indicators, over the work reported in [13]. *BEFs F9* through *F13* achieve the best ET_{hw} improvement ratios that reached 23300, 11233, and 5478 times better than the work reported in [13] for *F13* (See Fig. 7). In terms of TH_{hw}, functions *F4* and *F9* through *F13* significantly outperform the implementation in [13] with a maximum speedup of 1777 for *F11* (See Fig. 8). The best *SR* improvement ratio is achieved when targeting *F11* with a maximum of 195 (See Fig. 9).

6.2.2. Sequential versus parallel hardware implementations

In [14], Calazan et al. present a parallel co-processor for *PSO* on *FPGAs*. In the proposed implementation, all-particles computa-

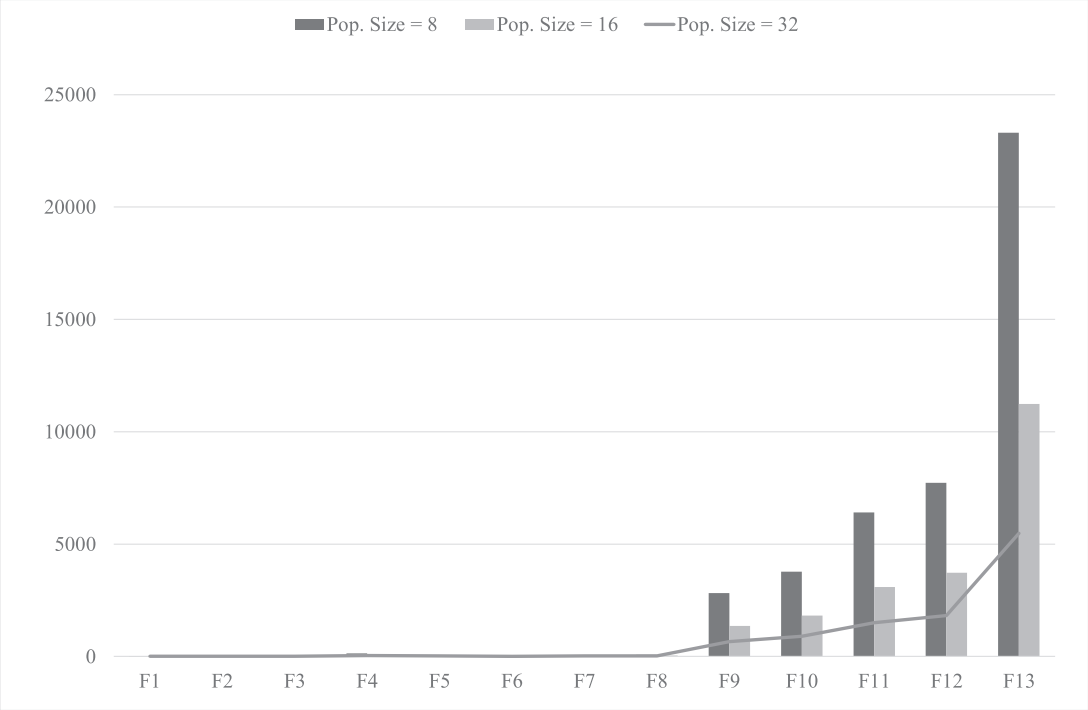


Fig. 7. *ETs* performance improvement ratio of the achieved execution time over the results reported in [13].

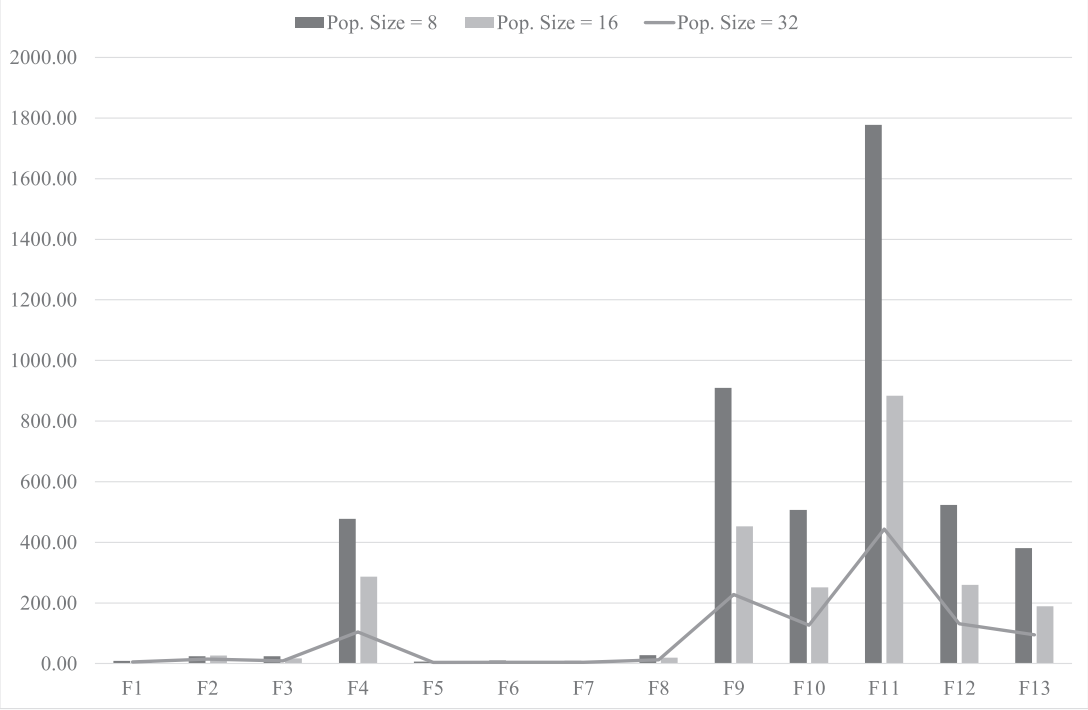


Fig. 8. *TH* speedup ratio of the achieved throughput over the results reported in [13].

tions are executed simultaneously until finding the g_{best}. To well-synchronize the computations, and prevent racing in the values of g_{best}, the velocity and position computations can only start once g_{best} is identified among all particles. The execution results are obtained under *Xilinx MicroBlaze* and a high-end *Virtex 6 FPGA*. In addition, the execution parameters are comparable to the ones adopted in this investigation; with the exception to that the authors did not specify the termination error threshold of the stop-page criterion. For a population size of 8, the reported *ETs* attain performance ratios that are between 2.4 and 67.27 times better than those achieved by the hardware implementation proposed in this investigation. However, the proposed implementation outperforms the reported *ET* under the *Xilinx MicroBlaze*; the performance improvement is between 3 to 85 times (See Fig. 10). *Virtex 6* is a high-end *FPGA* by *Xilinx*, while *Cyclone II* is presented as a low-end *FPGA* by *Altera*.

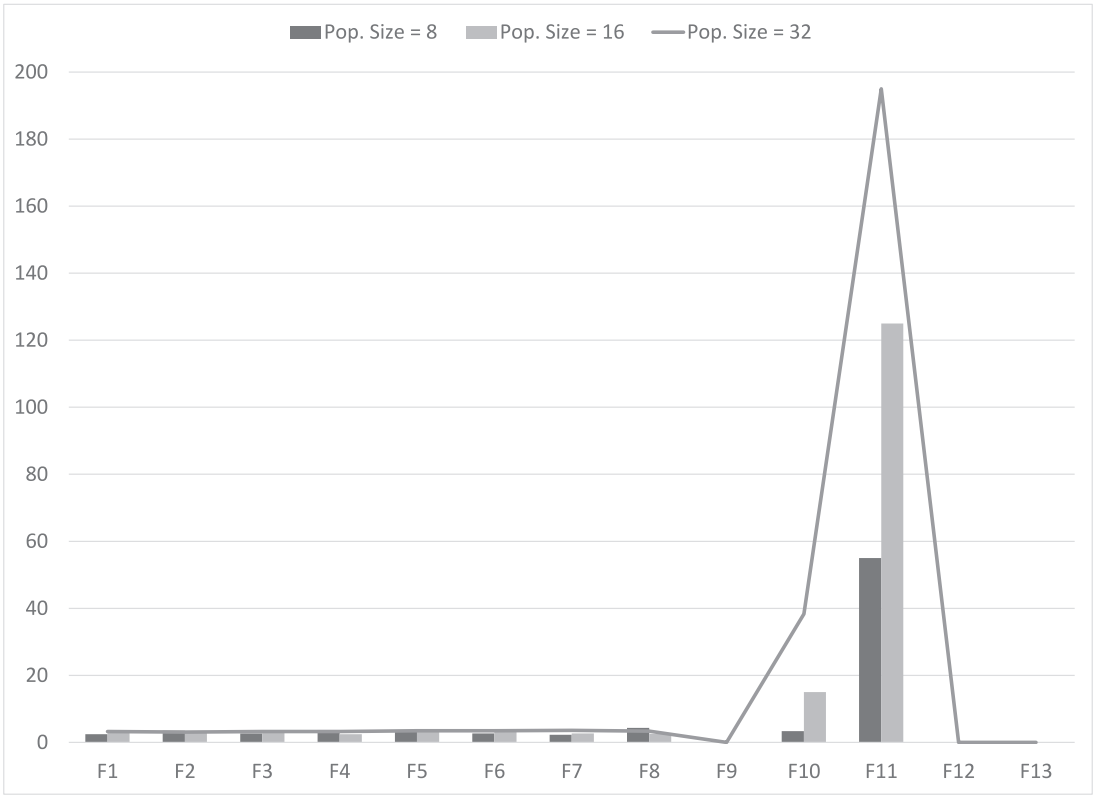


Fig. 9. *SR* improvement ratio of the achieved success rates over the results reported in [13].

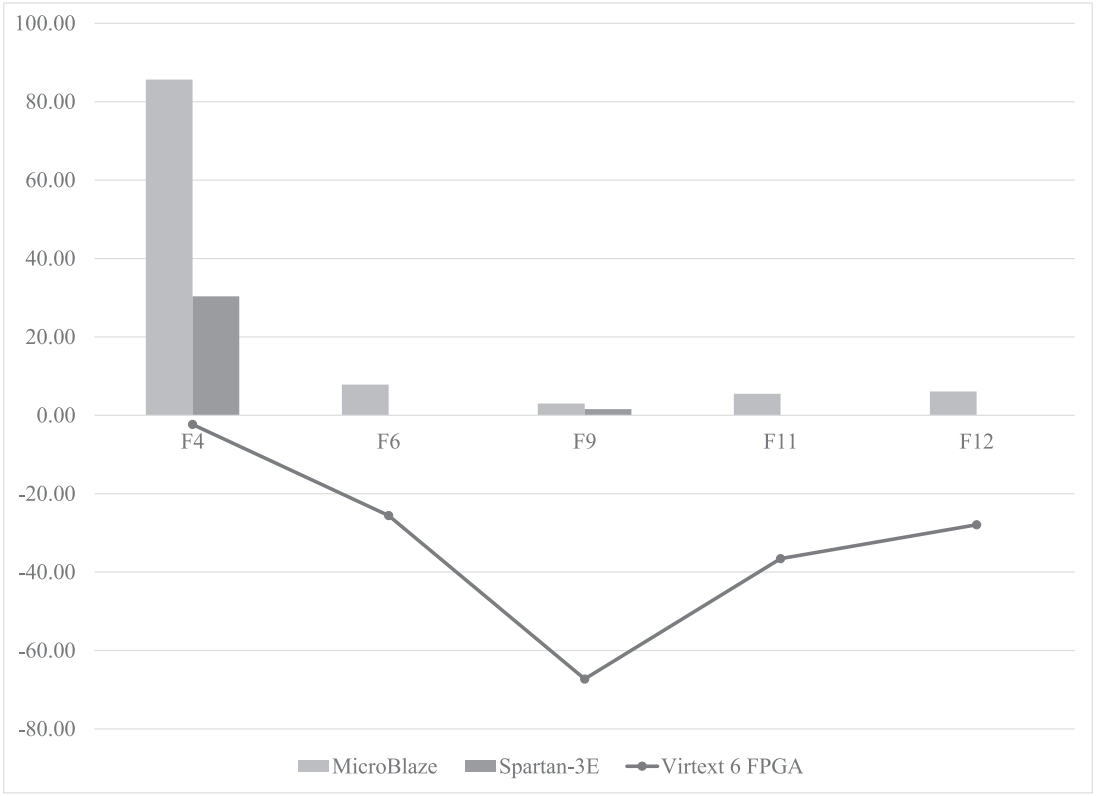


Fig. 10. Improvement ratios of the achieved *ET* in *HW* as compared with the results reported in [14]. The results achieved by the *Virtex 6 FPGA* are represented in the negative range as they outperform the *Cyclone II* implementation proposed in this investigation.

Table 10
General algorithmic profile.

BEF	BEFC	Mapped BEFC
F1	AQ	0.8
F3	AQ	0.8
F4	HQ	1
F5	HQ	1
F6	AQ	0.8
F8	AQ	0.8

Table 11
Software profile: *ET* and *TH*.

	ET(msec) per population size			TH(KBFEps) per population size		
BEF	8	16	32	8	16	32
F1	2.879	2.915	2.651	94.98	145.92	270.75
F3	2.377	2.285	1.96	85.07	142.60	265.31
F4	23.11	11.438	6.301	97.54	198.62	358.70
F5	2.402	2.033	1.845	60.88	115.14	210.73
F6	2.752	2.09	1.909	66.02	129.15	241.38
F8	106.86	8.015	6.199	8.35	144.75	253.87

Table 12
Software *CMIs:* SR_{sw} and PR_{sw}.

	SR_{sw}% per population size			PR_{sw} per population size		
BEF	8	16	32	8	16	32
F1	94	100	100	2.91	4.25	7.18
F3	87	99	100	2.32	3.29	5.20
F4	71	100	100	31.75	22.72	22.60
F5	100	100	100	1.46	2.34	3.89
F6	100	100	100	1.82	2.70	4.61
F8	60	100	100	14.87	11.60	15.74

In [23], Tewlode et al. present *HW* architectures that significantly accelerates execution performance of *PSO* over *SW* implementations. The proposed *HW* implementation targets a *Xilinx Spartan 3E FPGA*, while *SW* implementations are done using a *Xilinx MicroBlaze* and a Freescale *MC9S12DP256B* microcontroller. The authors also present a multi-swarm parallel *HW* implementation. The parallel implementation achieves a maximum speedup of 3.89 for *F4* and 6.96 for *F9* over the sequential *HW* implementation—with 30 particles divided among 5 sub-swarms. However, the *HW* implementation presented in this paper outperforms the sequential core in [23] by 30.34 times for *F4* and 1.59 times for *F9*.

6.3. Analysis of combined indicators

To enable the calculation of *OFI* as a main combined indicator, Tables 10 and 11 present the implementation results of the general algorithmic and *SW* profiles. Table 10 presents the complexity of the targeted *BEFs* that are needed for the *OFI* calculations and comparisons. Table 11 presents the *KI* results for the *SW* implementation. Moreover, Table 12 shows the measurements of the developed *CMIs* for the *SW* implementations. The calculated results of OFI_{hw}, OFI_{sw}, and *OFI* are shown in Figs. 11–13. The results confirm that *F4* attains the best OFI_{hw}, OFI_{sw}, and *OFI* ranking with values of 3.04, 3.15, and 3.04—for the three different population sizes of 8, 16, and 13. Accordingly, the proposed *PSO HW* and *SW* implementations can best perform when targeting the types of problems that are similar to *F4*.

Closely-related work in the literature, including [13–20,20–22], relies on an almost identical patterns of simple *KIs* to analyze and evaluate their implementations. The adopted analysis patterns can successfully classify implementations with-respect-to a single or a limited set of *KIs*. For example, the investigations in [13,14,20] successfully rank implementations per *ET*. However, combined indications and evaluative conclusions can hardly be made due to the

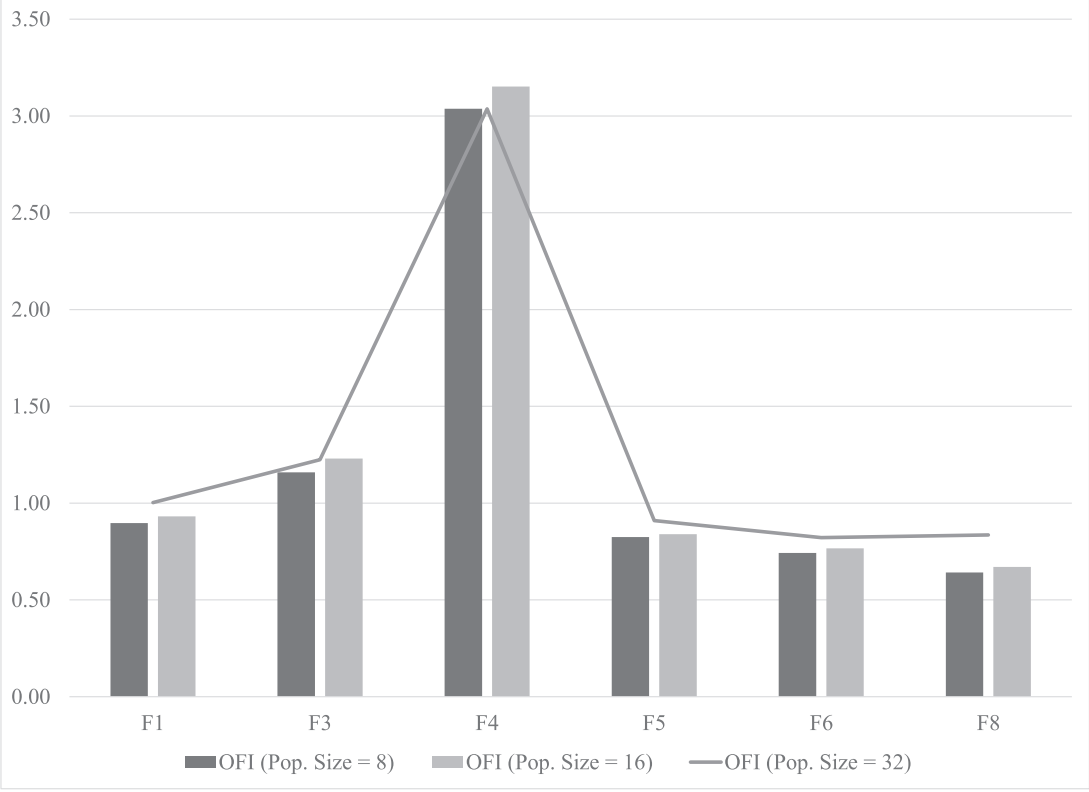


Fig. 11. The OFI_{hw} classification.

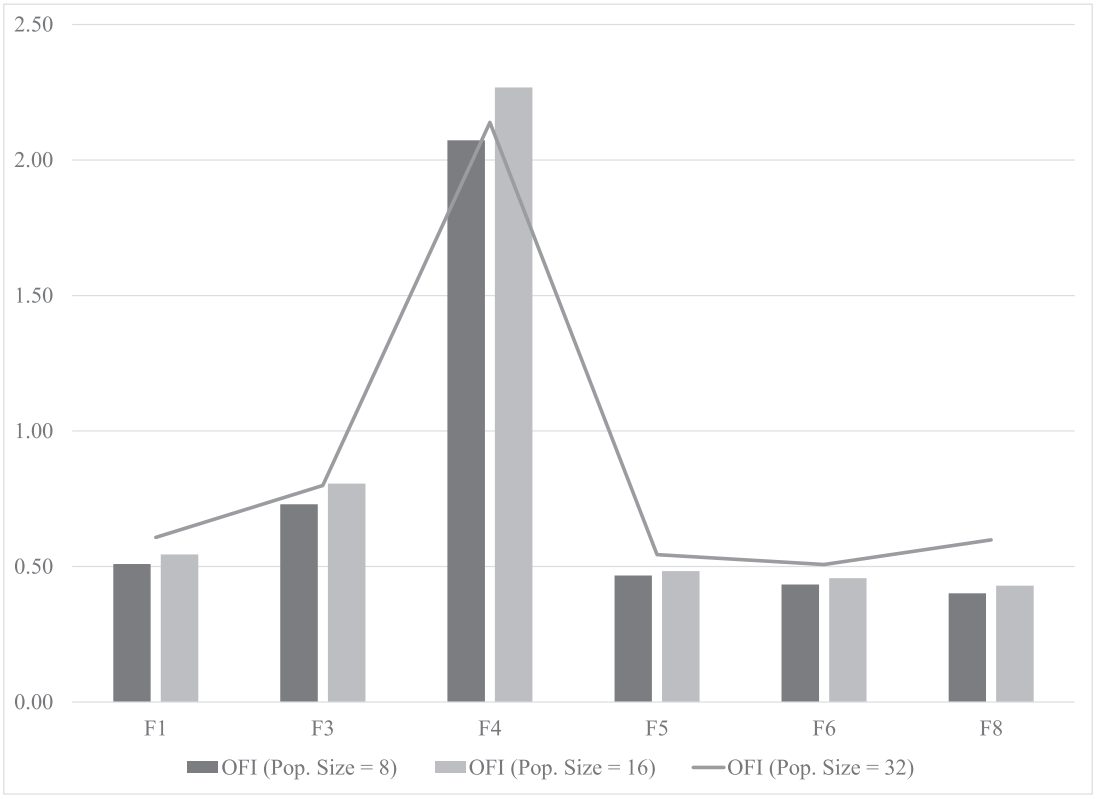


Fig. 12. The OFI_{sw} classification.

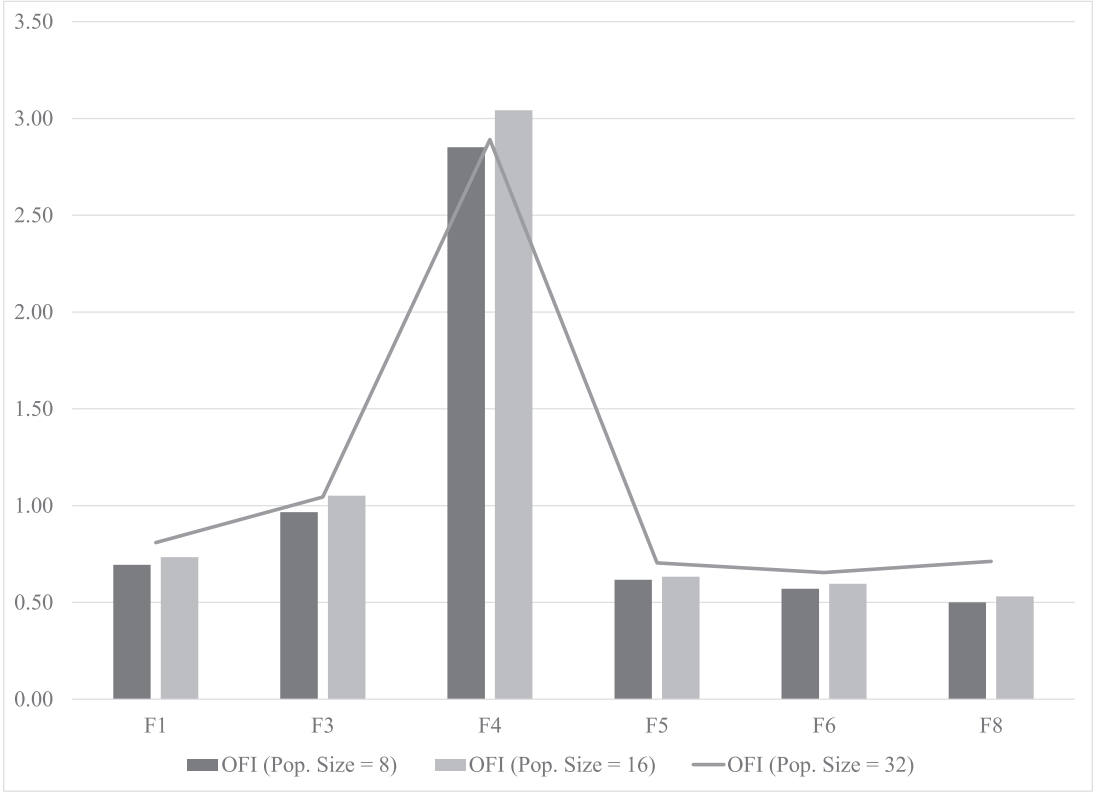


Fig. 13. The *OFI* classification.

absence of formal representations of *CMIs*. For instance, it is not straightforward to identify the evaluation function that can best perform in terms of a compromise that includes *ET, TH, SR*, etc. Our investigation confirms that classifications of implementations based on heterogeneous properties is limitedly addressed in the literature. With no doubt, the identification of the type of problem that can be effectively performed per algorithm implementation is an important evaluation. Such identifications can soundly identify suitable implementation options per application.

6.4. General evaluation

The current investigation can be evaluated as related to the developed hardware processor, and analysis framework development and application. The developed processor fully maps the *PSO* algorithm onto an *FPGA*. As compared with similar work, the proposed processor attains higher-speeds than similar implementations in the literature. In addition, the results show that, when using partitioned *HW* and *SW* implementations on *FPGA* boards, communication between the *FPGA* and other on-board components can significantly reduce the processing throughput.

The developed framework is unique in combining algorithmic, hardware, and software characteristics to provide unified performance evaluation criteria and useful performance indicators. The investigation proposes the creation of unified indicators that can capture specific qualities in terms of a wide range of heterogeneous key performance indicators, such as the *OFI*. The *OFI* serves as a master *CMI* while an indicator like the *BEFC* is developed with focus on evaluation function complexity. Indeed, the framework is scalable and upgradeable without changing the statistical computation or the structure of the measurement. For instance, an additional profile can be incorporated into the calculations of the *OFI* to include the performance characteristics of Graphics Processing Units.

At the application level, the developed framework can be used to examine qualities of importance and interest to optimization specialists. For example, the developed *OFI* successfully identifies the type of evaluation function that can be best solved by *PSO* and achieve the highest overall performance in terms of the identified *KIs*. The *OFI* enables classifying the *PSO* algorithm performance as targeting a heterogeneous set of evaluation functions. In addition, the framework produces a rich and comprehensive set of reference *KIs. KIs*, such as *ET, TH, LE*, and *LR*, are independent of the context of application and thus highly reusable. Other *KIs*, such as *NBFE, SR, PR*, and *SRD* are specific to optimization algorithms including the *PSO*.

The proposed framework aims at capturing the *HW* and *SW* properties. The current investigation doesn't include partitioned *HW* and *SW* implementations. However, partitioned implementations can be analyzed, based on the proposed framework, by enabling measurements of *KIs* under the *HW* and *SW* subsystems. The proposed partitioned *KI* measurements can capture subsystems' characteristics. In addition, well-defined *CMIs* can classify different partitioning strategies per optimization target, such as, area, speed, power consumption, etc.

7. Conclusion

Optimization is a key approach in engineering that enables effective solutions. *PSO* is a current and widely used heuristic; it is well-known for its effectiveness in application. In this paper, an on-chip *PSO* implementation is developed and mapped onto an *FPGA*. The developed processor significantly outperform the partitioned *HW* and *SW* implementations of [13] that target the same development board, *FPGA, BEFs*, and the execution parameters. Our proposed *HW* implementation achieves an ET_{hw} improvement ratio of 23,300 for *F13*, a TH_{hw} speedup of 1777 for *F11*, and the best *SR* improvement ratio of 195 for *F11* over the results reported in [13]. This paper includes the development of a statistical framework that enables thorough analysis and evaluation of optimization algorithms, such as *PSO*. The proposed indicators include *NBFE, BEFC, ET, TH, LE, LR, SR, PR, SRD,* OFI_{hw}, OFI_{sw}, and *OFI*. The analysis of results confirms that, when targeting *F4, PSO* achieves the highest performance characteristics with the highest *OFI* value of 3.15. The presented framework enjoys being reusable in the wider optimization context. Future work includes the development of pipelined, parallel, and multi-swarm *PSO* processors. The statistical framework can be expanded to capture partitioned implementations and to consider additional processing systems, such as Graphics Processing Units. Furthermore, the statistical framework can be applied to other metaheuristic algorithms for a wider study. The proposed framework is applicable outside the context of optimization [34,38].

Declaration of Competing Interest

Authors declare that they have no conflict of interest.

Appendix A

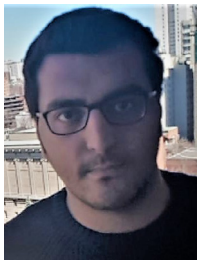
Acronym	Definition
BEF	Benchmark Evaluation Function
CMI	Combined Measurement Indicator
ET	Execution Time
FPGA	Field Programmable Gate Arrays
FSM	Finite State Machine
HW	Hardware
GBM	Generic Benchmark Mode
GAP	General Algorithmic Profile
HWP	Hardware Profile
KI	Key Indicator
BEFC	Benchmark Evaluation Function Complexity
LE	Logic Element
LR	Logic Register
LUT	Look-up Table
NBFE	Number of Benchmark Function Evaluations
OFI	Optimization Fitness Indicator
PSO	Particle Swarm Optimization
PR	Performance Rate
RNG	Random Number Generator
SR	Success Rate
SRD	Success Rate Density
SW	Software
SWP	Software Profile
TH	Throughput

References

[1] S.J. Kasbah, I.W. Damaj, R.A. Haraty, Multigrid solvers in reconfigurable hardware, J. Comput. Appl. Math. 213 (1) (2008) 79–94, doi:10.1016/j.cam.2006.12.031.
[2] S.J. Kasbah, I.W. Damaj, The Jacobi method in reconfigurable hardware, in: World Congress on Engineering, 2007, pp. 823–827.
[3] J.-S. Chou, A.-D. Pham, Nature-inspired metaheuristic optimization in least squares support vector regression for obtaining bridge scour information, Inf. Sci. 399 (2017) 64–80.
[4] M. El-Abd, Performance assessment of foraging algorithms vs. evolutionary algorithms, Inf. Sci. 182 (1) (2012) 243–263. Nature-Inspired Collective Intelligence in Theory and Practice. doi: 10.1016/j.ins.2011.09.005.
[5] M.E. Aydin, R. Kwan, J. Wu, Multiuser scheduling on the LTE downlink with meta-heuristic approaches, Phys. Commun. 9 (2013) 257–265, doi:10.1016/j.phycom.2012.01.004.
[6] M.E. Aydin, R. Kwan, C. Leung, C. Maple, J. Zhang, A hybrid swarm intelligence algorithm for multiuser scheduling in HSDPA, Appl. Soft Comput. 13 (5) (2013) 2990–2996, doi:10.1016/j.asoc.2011.12.007.
[7] M. Clerc, Particle Swarm Optimization, 93, John Wiley & Sons, 2010.
[8] J.L. Awange, B. Paláncz, R.H. Lewis, L. Völgyesi, Particle swarm optimization, in: Mathematical Geosciences, Springer, 2018, pp. 167–184.

[9] I. Damaj, J. Hawkins, A. Abdallah, Mapping high-level algorithms onto massively parallel reconfigurable hardware, in: IEEE International Conference of Computer Systems and Applications, 2003, pp. 14–22.
[10] S.M. Trimberger, Three ages of FPGAs: A retrospective on the first thirty years of FPGA technology, Proc. IEEE 103 (3) (2015) 318–331.
[11] J. de Fine Licht, M. Blott, T. Hoefler, Designing scalable FPGA architectures using high-level synthesis, in: Proceedings of the 23rd ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, ACM, 2018, pp. 403–404.
[12] X. Zou, L. Wang, Y. Tang, Y. Liu, S. Zhan, F. Tao, Parallel design of intelligent optimization algorithm based on FPGA, Int. J. Adv. Manuf. Technol. 94 (2018) 1–14.
[13] S.-A. Li, C.-C. Hsu, C.-C. Wong, C.-J. Yu, Hardware/software co-design for particle swarm optimization algorithm, Inf. Sci. 181 (20) (2011) 4582–4596. Special Issue on Interpretable Fuzzy Systems. doi: 10.1016/j.ins.2010.07.017.
[14] R.M. Calazan, N. Nedjah, L.M. Mourelle, A hardware accelerator for particle swarm optimization, Appl. Soft Comput. 14 (2014) 347–356, doi:10.1016/j.asoc.2012.12.034.
[15] G.S. Tewolde, D.M. Hanna, R.E. Haskell, A modular and efficient hardware architecture for particle swarm optimization algorithm, Microprocess. Microsyst. 36 (4) (2012) 289–302, doi:10.1016/j.micpro.2012.02.001.
[16] M.S. Ben Auemur, A. Sakly, FPGA implementation of parallel particle swarm optimization algorithm and compared with genetic algorithm, Int. J. Adv. Comput. Sci. Appl. 7 (8) (2016), doi:10.14569/IJACSA.2016.070809.
[17] C. Karakuzu, F. Karakaya, M.A. Çavuşlu, FPGA implementation of neuro-fuzzy system with improved PSO learning, Neural Netw. 79 (C) (2016) 128–140, doi:10.1016/j.neunet.2016.02.004.
[18] M.B. Abdelhalim, S.E.-D. Habib, An integrated high-level hardware/software partitioning methodology, Des. Autom. Embed. Syst. 15 (1) (2011) 19–50, doi:10.1007/s10617-010-9068-9.
[19] N. Nedjah, L. de Macedo Mourelle, A Reconfigurable Hardware for Particle Swarm Optimization, Springer International Publishing, Cham, pp. 29–42. doi:10.1007/978-3-319-03110-1_3.
[20] G.S. Tewolde, D.M. Hanna, R.E. Haskell, Accelerating the performance of particle swarm optimization for embedded applications, in: 2009 IEEE Congress on Evolutionary Computation, 2009, pp. 2294–2300, doi:10.1109/CEC.2009.4983226.
[21] X.-H. Yan, F.-Z. He, Y.-L. Chen, A novel hardware/software partitioning method based on position disturbed particle swarm optimization with invasive weed optimization, J. Comput. Sci. Technol. 32 (2) (2017) 340–355, doi:10.1007/s11390-017-1714-2.
[22] S.-A. Li, C.-C. Wong, C.-J. Yu, C.-C. Hsu, Hardware/software co-design for particle swarm optimization algorithm, in: 2010 IEEE International Conference on Systems, Man and Cybernetics, 2010, pp. 3762–3767, doi:10.1109/ICSMC.2010.5641826.
[23] G.S. Tewolde, D.M. Hanna, R.E. Haskell, Hardware PSO for sensor network applications, in: 2008 IEEE Swarm Intelligence Symposium, 2008, pp. 1–8, doi:10.1109/SIS.2008.4668308.
[24] M.B. Abdelhalim, A.E. Salama, S.E.-D. Habib, Constrained and unconstrained hardware-software partitioning using particle swarm optimization technique, in: A. Rettberg, M.C. Zanella, R. Dömer, A. Gerstlauer, F.J. Rammig (Eds.), Embedded System Design: Topics, Techniques and Trends, Springer US, Boston, MA, 2007, pp. 207–220.
[25] T.-Y. Lee, Y.-H. Fan, Y.-M. Cheng, C.-C. Tsai, R.-S. Hsiao, Enhancement of hardware-software partition for embedded multiprocessor FPGA systems, in: Intelligent Information Hiding and Multimedia Signal Processing, 2007. IIHMSP 2007. Third International Conference on, 1, IEEE, 2007, pp. 19–22.
[26] M. Ettouil, H. Smei, A. Jemai, Particle swarm optimization on FPGA, in: 2018 30th International Conference on Microelectronics (ICM), IEEE, 2018, pp. 32–35.
[27] M. Ettouil, H. Smei, A. Jemai, M. Ghazel, Codesign of an IoT using a metaheuristic IP, in: 2018 International Conference on Internet of Things, Embedded Systems and Communications (IINTEC), IEEE, 2018, pp. 153–157.
[28] T.L. Dang, Y. Hoshino, Hardware/software co-design for a neural network trained by particle swarm optimization algorithm, Neural Process. Lett. 49 (2) (2019) 481–505.
[29] A. Trimeche, A. Sakly, A. Mtibaa, Implementation of PSO algorithm for MIMO detection system in FPGA, Int. J. Electron. 105 (1) (2018) 42–57.
[30] I. Damaj, M. Imdoukh, R. Zantout, Parallel hardware for faster morphological analysis, J. King Saud Univ. Comput. Inf. Sci. 30 (4) (2018) 531–546, doi:10.1016/j.jksuci.2017.07.003.
[31] I.W. Damaj, Parallel algorithms development for programmable logic devices, Adv. Eng. Softw. 37 (9) (2006) 561–582.
[32] I. Damaj, High-Level Synthesis, Wiley, pp. 1–10. doi:10.1002/9780470050118.ecse177.
[33] B. Shackleford, M. Tanaka, R.J. Carter, G. Snider, FPGA implementation of neighborhood-of-four cellular automata random number generators, in: Proceedings of the 2002 ACM/SIGDA Tenth International Symposium on Field-programmable Gate Arrays, in: FPGA '02, ACM, New York, NY, USA, 2002, pp. 106–112, doi:10.1145/503048.503064.
[34] I. Damaj, S. Kasbah, An analysis framework for hardware and software implementations with applications from cryptography, Comput. Electr. Eng. 69 (2018) 572–584, doi:10.1016/j.compeleceng.2017.06.008.
[35] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, et al., Introduction to Algorithms, 2, MIT Press Cambridge, 2001.
[36] J.L. Hennessy, D.A. Patterson, Computer Architecture: A Quantitative Approach, Elsevier, 2011.
[37] D. Bishop, VHDL support library, 2008. Online; accessed 26 October 2019.
[38] I.W. Damaj, A.M. El Hajj, H.T. Mouftah, An analytical framework for effective joint scheduling over TDD-based mobile networks, IEEE Access 7 (2019) 144214–144229, doi:10.1109/ACCESS.2019.2945849.

Issam Damaj Ph.D ME BE SMIEEE MASEE, is an Associate Professor of Electrical and Computer Engineering (ECE) at Beirut Arab University (BAU). At BAU, he is the Director of the Center for Quality Assurance. Before joining BAU, he spent 13 years in professorial ranks in higher education institutions in Kuwait (American University of Kuwait, AUK, 10 years) and Oman (Dhofar University, DU, 3 years). During his tenure, he published 69 technical papers and 9 book chapters-in addition to various short papers and technical reports. His research interests include hardware design, smart cities, vehicular technology, and engineering education. He is an associate editor and a reviewer with publishers that include IEEE, Elsevier, and Springer. In addition, he is the recipient of various awards in mentoring, service, research, and academic high distinction. Dr. Damaj is a senior member of the IEEE. He maintains an academic website at https://www.idamaj.net.



Mohamed Elshafei, ME BE, is a Ph.D. student in Software Engineering and a research assistant at Data-driven Analysis of Software Laboratory, Concordia University, Quebec, Canada. He received a Bachelor of Engineering in Computer Engineering from American University of Kuwait in 2013. In 2016, he received a Master of Science in Computer Engineering from Kuwait University. His search interests include artificial intelligence and machine learning.

Mohammed El-Abd, Ph.D. ME BE SMIEEE, is an Associate Professor of Computer Engineering in the ECE Department at the American University of Kuwait (AUK). Dr. El-Abd has over 50 publications on the form of journal articles, book chapters, conference papers, and abstracts. His research interests include meta-heuristics, swarm and evolutionary intelligence, cooperative search, continuous and discrete optimization, large-scale optimization, robotics, and engineering education.

Mehmet Emin Aydin, Ph.D. ME BE, is a Senior Lecturer in Computer Science at the Computer Science and Creative, University of West England, Bristol, UK. Prior to this post, he worked in academic and research positions for various universities including University of Bedfordshire, London South Bank University and University of Aberdeen. He is an editorial board member of a number of international peer-reviewed journals and have been serving as committee member of various international conferences. His research interests include parallel and distributed metaheuristics, wired/wireless network planning and optimization, combinatorial optimization, evolutionary computation and intelligent agents and multi agent systems.