A Modified Binary Firefly Algorithm to Solve Hardware/Software Partitioning Problem

Mourad Khetatba and Rachid Boudour
Department of computer science, LASE Laboratory- Badji Mokhtar University, Annaba, Algeria
E-mail: khetatbam@yahoo.fr and racboudour@yahoo.fr

Keywords: hw/sw partitioning, co-design, Firefly, meta-heuristic

Received: January 9, 2021

Hardware/Software (Hw/Sw) partitioning is a crucial step in Hw/Sw co-design that determines which components of the embedded system could be implemented on hardware and which ones on software. It aims to find a design implementation that fulfills all the specification requirements (functionality, goals, and constraints) at a low cost. Most formulations of the Hw/Sw partitioning dilemma have proven to NP-hard optimization problems. The firefly algorithm (FA) emerges as a significant tool of Swarm Intelligence that has been applied in many areas of optimization. The main purpose of this paper is to present a modified binary firefly algorithm to solve Hw/Sw partitioning problems. We compare the performance and the quality of the solution of the proposed algorithm with two recently proposed FA variants namely the Naive Bayesian Binary Firefly Algorithm (NBBFA) and the Binary Firefly Algorithm (BFA); as well as other algorithms major partitioning in the literature. The computational results show that it produced better results than the all algorithms used.

Povzetek: V tem članku je predstavljena sprememba algoritma Firefly za reševanje težave s particioniranjem strojne in programske opreme.

1 Introduction

The embedded systems have become omnipresent in a wide variety of applications and typically consist of a combination of hardware components and one or more microprocessors executing software functionalities. Their complexity is increasing. Hence, the system designer has the difficult task of selecting the appropriate hardware/software components for building an embedded system for a given application, by satisfying certain constraints. They present colossal business opportunities whose limits are still far from being reached. Winning in the marketplace requires system development teams must be the first to put better products to the market by minimizing Time To Market (TTM). At the same time, reducing the cost of the product must be done. The need for co-designing hardware and software has long been pointed out for the development of those systems. The software is used for flexibility while specialized hardware delivers performance in the embedded domain. Hw/Sw co-design investigates the concurrent design of hardware and software components of complex electronic systems. It tries to exploit the synergy of hardware and software to optimize and/or satisfy design constraints [1]. A succession of steps starting with the specification of the system tasks to their synthesis forms what is called the Co-design process.

At the specification step, designing a complex system necessarily involves cutting down its behavior into a set of functions. This step is followed by another where to decide how its functions will be implemented. Hw/Sw partitioning is an important development step during HW/SW co-design. According to Vahid [2], Hw/Sw partitioning is the problem of defining what module of the system will be executed as a series of instructions (software) and what module will run in parallel circuits on some chip as FPGA (hardware), such as to achieve design goals like performance, cost, size, and power. Hence, its primary task is the division of full design into the hardware/software parts of the target structure while respecting all kinds of restrains, and provides the best compromising scheme in Hw/Sw partitioning.

The two hardware and software implementations typically have complementary advantages and disadvantages. Hardware-executed partitions usually perform faster at a cost of increased hardware area and higher power consumption, while software tasks are much easier to develop and modify, and they consume less power compared to the hardware partitions. Critical partitions of the system should be implemented in hardware, whereas the others in software. Finding an optimal partition is a tedious problem because of the large number and different characteristics of the specification of the functions that have to be considered. Hw/Sw partitioning problem to be solved can be expressed as an optimization problem that seeks to minimize one or more criteria by defining a cost function. It is considered as NP-hard problem for most cases [3, 4, 5]. The latter presents a formal definition in form of task graphs, widely used in partitioning representation.

To bypass these hardiest problems and still provide good solutions, particular studies were recently oriented

to meta-heuristic nature-inspired algorithms, which can be used to get high-quality results in a reasonable time and with small computational efforts, even if they do not guarantee to obtain globally optimum solutions. Meta-heuristic algorithms including evolutionary and swarm intelligence algorithms have shown successful results when solving constrained problems [6, 7].

One of the recent swarm intelligence algorithms is Firefly algorithm (FA) that was developed by [6]. This algorithm is inspired by the social behavior of fireflies, mating, and exchange of information using light flashes. Owing to its few parameters to adjust, it is easy to understand, to realize, and to compute. Simulations and results indicate that FA is superior to PSO, GA, and ACO taking in account both efficiency and success rate in solving continuous optimization problems [8, 9]. Several researchers have improved the standard FA by modifying the control and attractiveness parameters or by hybridizing with other meta-heuristics [10]. Thus, many FA variants have been developed to solve various optimization problems such as Robotics [11], Civil engineering [12], and Chemistry [13]. A list of other different optimization problems solved by FA can be found in [14].

As mentioned above, the Firefly Algorithm (FA) is a nature-inspired optimization algorithm that can be successfully applied to continuous optimization problems. However, a lot of practical problems are formulated as discrete optimization problems and the algorithm cannot be applied directly to these discrete problems. But, the results produced by the FA algorithm in solving discrete NP-hard problems such as image compression and processing [15], shape and size optimization [16], and manufacturing cell problem [17] encourage researchers to design novel FAs for discrete optimization problems.

To solve the permutation flow shop scheduling problems, a discrete firefly algorithm for minimizing the makespan was proposed by Sayadi et al. [18], which was designed by modifying the basic firefly algorithm to adapt to solving discrete problems. A modified version of FA was used by Durkota [19] to solve the class of discrete problems called Quadratic Assignment Problems (QAP), where a mapping into newly developed discrete functions of continuous functions such as attraction, distance, and movement, is recommended. A binary FA is proposed by Palit et al [20] to deduce the meaning of an encrypted message for cryptanalysis. Another researcher, Falcon et al. [21] presented a binary adaptive Firefly Algorithm for fault identification in parallel and distributed system by using binary encoding on candidate solution with adaptive light absorption coefficient to improve the search. Chandrasekaran [22] developed a binary version of FA to solve the reliability constrained unit commitment problem. Khadwilard et al. [23] solved the job shop scheduling problems using the Firefly algorithm. To tackle the mapping from a continuous search space to discrete search space for solving the non-unicost set covering problem which is a well-known NP-hard discrete optimization problem, Crawford et al. [24] proposed a binary coded firefly algorithm based on the use of different transfer functions investigated in terms of convergence speed and accuracy of results. A discrete Firefly algorithm (DFA) combined with the local search (LS) method to enhance the searching accuracy and information sharing among fireflies was proposed by Karthikeyan [25] for solving multi-objective flexible job-shop scheduling problems. Najeeb et al. [26] presented all steps of applying Firefly algorithm to Constraint Satisfaction Problems (CSPs) which have discrete nature. This novel feature selection method is based on the binary Firefly Algorithm (FA) and Naïve Bayesian Classifier (NBC). A Naive Bayesian Binary Firefly Algorithm (NBBFA) was developed by Rajalaxmi et al. [27]. Each firefly moves to find the optimal gene set from the search space for cancer identification based on the fitness evaluation done through the naïve Bayes classifier. A Novel Binary Firefly Algorithm for the Minimum Labeling Spanning Tree Problem was presented by M. Lin et al. [28]. This novel method allows the updating positions of fireflies, which makes the algorithm more suitable for solving discrete problems.

In most works above, the updating procedure of the standard Firefly algorithm will be used and the result will be converted to discrete values. The position of Firefly shifts between ‘‘1’’ and ‘‘0’’ in discrete space. To accomplish this, they used the sigmoid function to constrain the position value of each firefly to the interval [0, 1]. A complete survey for updating the position of fireflies to transform continuous variables to binary variables is presented by Tilahun et al. [29]. Some variants of the sigmoid functions, S-shaped functions, some variants of the tan hyperbolic functions, and V-shaped functions are given. More generally, for a complete overview of the topic, several reviews were carried out. For instance, we highlight the recent works by [30].

According to the best of our knowledge, there is no published work dealing with the Hw/Sw partitioning problems by using FA Algorithm. Thus, in this paper, we proposed a modified binary FA algorithm to solve those problems. Our approach takes into account several constraints such as available area, execution time, and memory. Optimal partitioning solutions are obtained via this algorithm which is considered a new method in this field.

The rest of the paper is organized as follows. Section 2 lists the most reputed work in the field of Hw/Sw partitioning methods. Section 3 illustrates briefly the Firefly algorithm (FA). The partitioning problem formulation is given in section 4. Experiment results are discussed in Section 5. The work is concluded and perspectives to future work are given in Section 6.

2 Related works

The earliest works for the Hw/Sw partitioning problem can be found in [31]. Traditionally, Hw/Sw partitioning has been done manually, causing substantial delay [32]. The system designer decided which functional objects (or basic blocks) of the system could be implemented in hardware and which ones could be realized in software,

considering his experience in the field. These manual approaches directly affect the development time and the quality of the selected solution. Therefore, they were limited only to small designs with small number of constituent blocks [32]. But, the complexity of the embedded system keeps increasing; thus, an efficient HW/SW partitioning technique is required to ensure a cost-effective embedded system while performance constraints are satisfied.

Using automatic Hw/Sw partitioning has become a necessity, and the partitioning results directly affect the system performance. Enormous difficulties may be encountered in obtaining the optimal solution since partitioning is considered a combinatorial optimization problem. Over the last two decades, a wide range of approaches has been proposed: (a) start from the pure software functional specification of the problem, then migrate critical software functions to a hardware implementation [33], Hardware-executed functions usually perform faster with more expensive cost, and (b) start with a pure hardware functional specification of the system and iteratively moves the non-critical parts or functions of the problem to the software as long as performance constraints are fulfilled[34]. Software function implementation requires more flexibility and less cost, but more execution time.

Hw/Sw partitioning is considered a combinatorial optimization problem (COP). Due to the complexity of this category of problems, there are two kinds of automatic approaches solving such problems; namely exact methods and approached ones. Exact methods are simple to implement and allow the exploration of all possible configurations; therefore, they guarantee to provide optimal solutions, respecting all constraints and requirements. However, these algorithms become intractable when the problem size is large.

Since Hw/Sw partitioning is considered NP-hard in more cases [3], the solution space is immense, so it is impossible to get an exact solution in a reasonable amount of time. The idea is to find good-quality solutions without investigating the entire search space. A variety of "natural phenomena" such as evolution imitation, annealing, or knapsack packing help to solve the NP-hard problem of Hw/Sw partitioning by using Meta-heuristic algorithms [35]. Their use in many applications shows their efficiency and effectiveness to solve large and complex problems.

In literature, meta-heuristics methods are divided into two classes: firstly, meta-heuristics based on a single solution such as Simulated Annealing (SA)[36] and Tabu search(TS) [37], and secondly, meta-heuristics based on a population of solutions. These meta-heuristics start from an initial population of solutions often generated randomly. Then, they iteratively apply the generation of a new population and the replacement of the current population. This process iterates until a given stopping criterion. The performance of the population-based algorithms is measured by checking their ability to establish a proper compromise between the two concepts exploration and exploitation. Avoiding getting trapped in local optima and premature convergence, meta-heuristics must have a balance between these two concepts mentioned previously [38].

Many approaches focus on algorithmic aspects since the HW/SW partitioning is proven as an NP-hard optimization problem. Therefore, for a large-scale partitioning problem, researchers have applied many heuristic algorithms to HW/SW partitioning. In the literature, we can cite some extensively used and popular meta-heuristics algorithms for solving those problems such as simulated annealing algorithms [36], Tabu search [37], genetic algorithms [39,40], particle swarm optimization [41,42], ACO[43,44], differential evolution DE[45] and Bat Algorithm[46].

Other designers seek to obtain more optimal partitioning solutions by focusing on a combination of existing meta-heuristic algorithms to solve the HW/SW partitioning problems. Iguider et al. [47] combined the Lagrangian Relaxation (LR) method with the 0–1 Knapsack Algorithm and the Genetic Algorithm to deal with the Hw/Sw problem, considering three metrics: hardware cost, execution time, and power consumption. The objective is to minimize one metric, yet respecting the constraints on the other two metrics. Yan et al. [48] developed a novel HW/SW partitioning method based on position disturbed particle swarm optimization with invasive weed optimization (PDPSO-IWO). To solve the premature convergence and avoid falling into local optima, the particles in PDPSO-IWO move away from the worst particle in the population, near which there is a potential predatory threat. Shi et al.[49] proposed three algorithms for multiple-choice hardware-software partitioning to minimize execution time and power consumption while meeting area constraints. To rapidly generate approximate solutions, Firstly, a heuristic algorithm was applied. Then, a customized tabu search algorithm can further refine the approximate solution. Finally, the exact solution was calculated by applying a dynamic programming algorithm Li et al. [50] combine the GA and TS algorithm which can be applied to the dynamically reconfigurable system. The experiment results have shown that their approach is a method with high performance and can map the task graph to a reconfigurable system with efficiency. To minimize the logic area of System on a Programmable Chip (SOPC) while respecting a time constraint, Dimassi et al.[51] incorporated the binary search trees(BST) into the genetic algorithm to address the problem of software/hardware partitioning. An et al. [52] combined the GA and the PSO algorithms. The solutions obtained by these combinations are more accurate than those given by classical algorithms in terms of cost and execution time metrics. In Ref. 53, a hybrid method of PSO and TS was proposed to solve the HW/SW partitioning problems. A combination of the clustering algorithm and the genetic algorithm was developed by Weijia et al. [54]. Experiment results have shown that this approach can accelerate converge to an appropriate solution of a complex system with more tasks. Zhao et al. [55] incorporate the simulated annealing algorithm in the genetic algorithm. Experiment results have proven that the proposed algorithm has given more accurate partitions than the original genetic algorithm.

3 Firefly algorithm

Swarm intelligence is akin to a collective form of intelligence seen in numerous animal species (insects, fish, birds, mammals,...), most of them exhibiting very developed social behaviors and important capacities of adaptation to their environment. Computing was indeed inspired by studies on swarm intelligence biological to produce innovative algorithms. Firefly algorithm is a relatively new swarm intelligence optimization method that was developed by Yang [6]. This algorithm is inspired by the social behavior of fireflies, mating, and the exchange of information using light flashes. As a reminder, Fireflies are small winged beetles capable of producing a cool flashing light to mutual attraction. This chemical light is generated from the lower abdomen of the bodies of these insects. The color of this light can be yellow, green, or pale red, with a wavelength between 510 and 670 nanometers. Females' fireflies can imitate the light signals of other species to attract the males' fireflies which they capture and devour. Thus, Fireflies communicate and attract each other with varied flashing patterns. Firefly algorithm idealizes some characteristics of the firefly behavior. They follow three rules [56]:

- All fireflies are unisex so that one firefly will be attracted to other fireflies regardless of their sex;
- Each firefly is attracted only to the fireflies, that are brighter than itself; the strength of the attractiveness is proportional to the firefly's brightness, which attenuates over the distance. The attractiveness decreases as the distance increases between two fireflies. If there is no brighter one than a particular firefly, the brightest firefly moves randomly,
- Brightness of every firefly determines its quality of solution; in most cases, it can be proportional to the objective function.

The objective function to be optimized must associate these phenomena. The Firefly algorithm is based on two important concepts: the light intensity variation and the attraction formulation. To simplify, the attraction of fireflies is determined depending on the brightness, where the brightness is determined with the function objective. The basic steps of the FA are summarized by the pseudo-code shown in Fig. 1, which considers the three rules discussed above.

According to recent research, simulation results for finding the global optima in solving optimization problems show that the firefly algorithm is superior to

```
1. Initialize the parameters:
   NP // Population size
   D  //  Problem dimension
   Max_gen // Maximum of iterations
2. Define objective function f(x), X=(x1,x2,…,xD)
3. Generate initial population of fireflies Xi(i=1,2,..NP)
4. Define light intensity Ii at Xi
5. Set light absorption coefficient
6. Set initial attractiveness
7. Set t at 0 // t is counter iteration
8. While(t max_gen)
9.    For (i=1 to NP ) // all fireflies
10.      For (j=1to NP) // all fireflies
11.         If( Ii  > Ij) Move firefly I towards j endif
12.         Update attractiveness and light intensity I
13.         Evaluate new solutions
14.      Endfor
15.   Endfor
16.   Rank the fireflies and find the current best solution
17.   Increment t
18. Endwhile
19. Display the best solution
```

Figure 1: FA algorithm pseudo-code.

both PSO and GA taking into account both efficiency and success rate [56, 57]. These facts give the inspiration to investigate to find optimal solutions using FA in solving Hw/Sw partitioning problems.

To design FA properly, the variation of light intensity and formulation of the attractiveness [4] must be defined. The light intensity varies with the distance exponentially and monotonically. It can be approximated as follows:

$$I = I_0 e^{-\gamma r^2} \quad (1)$$

Where I_0 represents the original light intensity and γ is a fixed light absorption coefficient. r_{ij} indicates the distance between firefly i and firefly j, at positions x_i and x_j, respectively, and is defined as follows:

$$r_{ij} = \sqrt{\sum_{k=1}^{d} (x_{i,k} - x_{j,k})^2} \quad (2)$$

Where x_i,k is the kth component of the spatial coordinate x_i of the ith firefly, and d is the number of dimensions.

The attractiveness β of fireflies is proportional to their light intensities *I*. It implies how strong it attracts other members of the swarm. The attractiveness β is expressed as follows:

$$\beta = \beta_0 e^{-\mu r^2} \quad (3)$$

Where r is the distance between two fireflies, β_0 is the attractiveness at r=0 and γ is a fixed light absorption coefficient.

The movement of a firefly i, which is attracted by a more attractive (i.e. brighter) firefly j, is given by the following equation:

$$x_i^{(t+1)} = x_i^{(t)} + \beta_0 e^{-\mu r^2} \left(x_j^{(t)} - x_i^{(t)}\right) + \alpha \left(rand - \frac{1}{2}\right) \quad (4)$$

Where the first term is the current position of a firefly, the second term is, attraction to another more attractive firefly, and the third term is randomization, with α being the randomization parameter, while rand is a randomly generated number from interval [0, 1].

4 Problem formulation

Today, the embedded system to be partitioned is modeled as a Directed Acyclic Graph (DAG), which becomes the input to the Hardware/Software (HW/SW) partitioning step. In this work, the HW/SW partitioning problem is based on the same system model which is used in [58, 3]. The node in the DAG stands for a basic block; the edges stand for communication and precedence relationship between blocks.

The DAG G = (V, E), which is used to describe the system behavior, consists of a set of functions or tasks which are represented by vertices V = {v_i | i = 1,2,..N}, and a set of data and control dependencies which are represented by edges E = {e_{ij} | e_{ij} = (v_i,v_j), v_i, v_j εV}. Assume that N denotes the number of nodes(tasks), let X=(x_1,x_2,…,x_N) be the feasible solution set of the partitioning problem, where x_i ε {0,1} and x_i denotes how v_i(task i) is realized where x_i=1(x_i=0) means v_i is realized through hardware(software), m1 nodes which will be implemented in Hw and m2 others nodes in Sw(n=m1+m2),V^{Hw} = {v_i | i = 1,2,..m1},V^{Sw}= {v_i | i = 1,2,..m2}).

Clearly, this step of Hw/Sw partitioning has a dramatic impact on the cost and performance of the whole system. Some design-quality attributes which must perfectly describe the solution are used to measure the validity of the solution. Most works, in literature, were proposed to study the Hw/Sw partitioning problem with three metrics, execution time, hardware area, and communication cost[3,58,59,60]. Other approaches have also added to different cost metrics the power consumption and software memory usage [42,61,62,63].

In this article, a basic block or task can be defined as a 9-tuple vi =(T_i^{Sw},T_i^{Hw},A_i^{Hw}, Dm_i^{Sw}, Cm_i^{Sw}, Sc_i^{Sw}, Hc_i^{Hw}, Pc_i^{Hw}, Pc_i^{Sw}) where:

- T_i^{Sw} represents the execution time of the ith task if implemented in software processor,
- T_i^{Hw} denotes the execution time taken by the task i when executed in hardware
- The hardware implementation of the ith task requires area A_i^{Hw} on the hardware task.
- Software area DM_i^{Sw} (for data) and CM_i^{Sw} (for instructions) represent the software memory utilized by the ith module
- S_i^{Sw} and S_i^{Hw} represent the cost taken by the block i when implemented in software module and onto hardware
- Pc_i^{Sw} and Pc_i^{Hw} are respectively the power consumption in software and hardware of the ith block

On the other hand, each edge, e_i includes $Comc_{ij}$ which represents the communication cost between tasks v_i and v_j. An important assumption is made that vertices mapped onto the same computing unit have negligible communication latency; i.e. Sw-Sw or Hw-Hw communication Cost can be considered 0 for practical purposes. The communication cost in this context refers to the delay time required to transfer the data from the hardware module to the software module and vice versa.

The assignment of Hw/Sw partitioning is to map Sw tasks to CPU and Hw tasks to hardware components while satisfying design constraints. In this paper, the problem of HW/SW partitioning is formulated as a single objective optimization problem combining multiple cost terms into a single scalar function. The objectives used to guide the Hw/Sw partitioning algorithm through the optimization process are the total execution time (Texe), total hardware area(A), total memory required (M), total power consumption(Pc), and total global cost (Gc) calculated using the following equations, respectively.

$$T^{sw} = \sum_{i=1}^{n} \left((1 - x_i) * T_i^{Sw}\right) \quad (5)$$

$$T^{Hw} = \sum_{i=1}^{n} \left(x_i * T_i^{Hw}\right) \quad (6)$$

$$T_{exe} = T^{sw} + T^{Hw} \quad (7)$$

$$A = \sum_{i=1}^{n} (x_i * A_i^{Hw}) \quad (8)$$

$$M = \sum_{i=1}^{n} \left((1 - x_i) * \left(DM_i^{Sw} + CM_i^{Sw}\right)\right) \quad (9)$$

$$Gcom = \sum_{i=1}^{n} \left(\sum_{j \in V^{Sw}} (x_i * Ccom_{ij}) + \sum_{j \in V^{Hw}} \left((1 - x_i) * Ccom_{ij}\right) \right) \quad (10)$$

$$Pc = \sum_{i=1}^{n} \left((x_i * Pc_i^{Hw}) + \left((1 - x_i) * Pc_i^{Sw}\right) \right) \quad (11)$$

$$Gc = \sum_{i=1}^{n} \left((x_i * S_i^{Hw}) + \left((1 - x_i) * S_i^{Sw}\right) \right) + Gcom + Pc \quad (12)$$

Then, the cost function which expresses the factors that the designer wants to minimize and satisfying some constraints can be formulated as follows:

$$f(x_1, x_2, x_3, x_4) = \omega_1 * T_{exe} + \omega_2 * A + \omega_3 * M + \omega_4 * Gc \quad (13)$$

Where ω1, ω2, ω3, and ω4 are positive integers which reflect how much weightage is given to each associated metric for a particular partition. For example, to give more importance to execution time for the partitioning decision, the weight corresponding to the execution time metric can be increased and to ignore any metric included in cost function f(x1, x2, x3, x4), the weight corresponding to this metric must be set to zero.

Tmax, Amax, Mmax, Ccommax, Pcmax, and Gcmax represent execution time constraint, hardware area constraint, Communication constraint, Maximum power consumption required, Maximum Memory required, and the desired global cost respectively. For our study, the partitioning problem that consists of minimizing the objective function f(x1, x2, x3, x4) with the respect of constraints, can be formulated as the follows:

$$Minimize\ f(x_1, x_2, x_3, x_4)$$

$$Subject\ to \begin{cases} T_{exe} \leq Tmax \\ A \leq Amax \\ Gc \leq Gmax \end{cases} \quad (14)$$

Thus, the original firefly algorithm needs to be modified in the context of Hw/Sw partitioning problems. So, when firefly i moves towards firefly j, the position of firefly i must replace the real number by a binary number. For this purpose, we suggest to incorporate the mutation operator of the standard DE into FA Algorithm. Since the standard mutant operator generates real-coded vectors, not bit strings, a new probability estimation operator must be used to tackle this problem in the Binary Firefly algorithm. The probability estimation operator can effectively preserve the diversity of the population and enhance the global search ability. For more details, the probability estimation operator is defined by formula [(17),(18)] as follows:

$$P\left(x_{ij}^{(t+1)}\right) = \frac{1}{\left(1 + e^{-\frac{2b(MO - 0{,}5)}{1+2F}}\right)} \quad (17)$$

$$MO = x_{r1,j}^{(t)} + F * \left(x_{r2,j}^{(t)} - x_{r3,j}^{(t)}\right) \quad (18)$$

```
11. if (Ii>Ij)
      { Move firefly I towards firefly j by applying eq. (4);
        Update position x_i^(t+1) by applying eq. (19)
      }// endif
```

Figure 2: Using PEO in firefly algorithm.

Where b is a bandwidth factor used to increase the search efficiency, F is the scaling factor which is a positive constant; t is the index of generation; $x_{r1,j}^{(t)}$,$x_{r2,j}^{(t)}$ and $x_{r3,j}^{(t)}$ are the jth-bits of three randomly selected fireflies with index r1≠r2≠r3≠i.

By this scheme, three-parent fireflies will be considered aiming to establish the probability distribution model. The bit of the mutant firefly will then be "1" or "0". Hence, using the machine operator (MO) formula [(17), (18)] guarantees to provide a binary code for updating $x_i^{(t+1)}$. Once, the probability estimation vector is determined, the corresponding updated position of the firefly i, $x_i^{(t+1)}$, is deduced by applying the following equation:

$$x_{ij}^{(t+1)} = \begin{cases} 1 & if\ rand_{ij} < P(x_{ij}^{(t+1)}) \\ 0 & otherwise \end{cases} \quad (19)$$

Where $rand_{ij}$ is a stochastic number uniformly distributed within [0, 1] and $P(x_{ij}^{(t+1)})$ is the jth component of the probability vector of the ith firefly.

Thus, line 11 of the pseudo-code of the firefly algorithm given in figure 1 becomes as follows (figure 3).

5 Experiments and results

Important work has been done in hardware/software partitioning in recent years. Nevertheless, it is impossible to perform a comprehensive comparison of all the existing approaches due to the large incompatibility in their co-design environments and the lack of proper benchmarks [64]. Presently, there isn't a test set accepted by embedded system's Hw/Sw partitioning worldwide [65]. No attempts have been made to solve Hw/Sw partitioning problems by using the Firefly algorithm.

For our study, similar to the analysis methods in [47, 54, 59, 65, 66, 67], the instances of all metrics to be employed in the experiments, are generated randomly. The values of software execution time (T_i^{Sw}), hardware execution time (T_i^{Hw}), and hardware area occupied (A_i^{Hw}) were generated randomly, as in [47, 54, 59, 65, 66, 67]. For example, [65] used random values for the two metrics area and hardware execution time generated respectively in [0, 100] and [0, 60] to validate their results. For the case of hardware execution time (T_i^{Hw}), values were generated randomly in the range [1,50]; software execution time (T_i^{Sw}), values were generated randomly in the range [1,5* T_i^{Hw}]; while for the hardware area (A_i^{Hw}) values were generated in the range [1,60]. Table 1 summarizes the interval values for all metrics.

In order to evaluate the efficiency and performance of the proposed algorithm, we proposed, ten DAGs (Direct Acyclic Graph) are randomly generated using the TGFF (Task Graphs for free) tool [68] by specifying the number of nodes, every node is associated with one basic

Parameters	Value
hardware execution time (T_i^{Hw})	[1..50];
software execution time (T_i^{Sw})	5* T_i^{Hw}
hardware area (A_i^{Hw})	[1..60]
Memoy required (DM_i^{Sw}+CM_i^{Sw})	[1..100]
Hardware cost(S_i^{Hw})	[1..100]
Software cost(S_i^{Sw})	[1..20]
Communication cost($Ccom_{ij}$)	[1..30]
Hardware Power cost(Pc_i^{Hw})	[1..20]
Software power cost(Pc_i^{Sw})	[1..5]

Table 1: Metrics settings.

Algorithm	Control Parameters
BFA[28]	β0=1, γ =1, α=081
NBBFA[27]	β0=0,97, γ =1, α=0,2
BFA-PEO	F=0.2; b=20, β0=1, γ=1, α=0,81
GA	crossover = 0.8; Mutation = 0.05
BPSO	W=1,c1=c2=2;
saBDE[68]	crossover=0.8
DDE[69]	Mutation=0.05,crossover=0.8
BDE[69]	Mutation=0.05,perturbation=0.5
NBDE[70]	crossover=0.8
NMBDE[71]	Cr=0.8,f=1

Table 2: Parameters settings of each method.

block. In our study, then we set different metrics (area, software execution time, hardware execution time, etc.) for different blocks, and eventually, we get 10 DAGs with 100, 200, 300,400,500,600,700,800,900, and 1000 nodes respectively.

In this section, we compare the solution quality and performance of the proposed algorithm BFA-PEO with two recently proposed FA variants and other major partitioning algorithms in the literature. The involved algorithms are listed in Table 2. The population size of all algorithms was set as NP =30, the maximal generation number was 300. Table 2 lists the parameter settings of each meta-heuristic used in this comparison.

Two experiments were performed to verify and compare the effectiveness and the performance of the proposed algorithm BFA-PEO. We ran the algorithms 50 times for the same input by using the same population initialization and took the best of the cost function.

5.1 First experiment

We consider the problem that consists of minimizing the objective function f1(x1,x2) with the respect of the constraints on the execution time(Tmax) and hardware area(Amax). The goal is to find the optimal Hw/Sw partitioning with lower execution time and area. f1(x1,x2) can be expressed by using the following equation:

$$Minimize\ f1(x_1, x_2) = T_{exe} + A$$

$$Subject\ to$$

$$\begin{cases} T_{exe} \leq Tmax \\ A \leq Amax \end{cases}$$

Table 3 presents the experimental results achieved by the algorithms retained to conduct this study where the best fitness value is displayed. The best results of the algorithms are written in bold. Looking at table 3 and table 4, it is obvious that BFA-PEO searches out better solutions than the other algorithms while BFA and NBDE perform second-best results. The saBDE algorithm comes last, providing the worst results.

Consider an application specified as a DAG (directed acyclic graph) with 500 nodes. We remember that the candidate partitioning solutions are also vectors of 500 values set to 0 and 1 where each value of the vector is a module implemented in hardware (Hw) if it is equal to 1, or in software (Sw) if it is equal to 0. The best solution in this first experiment is a vector that contains the minimum of the execution time and area parameters,
The cost of the initial population is thus calculated using the objective function. We obtain the best initial solution composed of 239 software modules and 261 hardware modules with fitness equal to 41897. BFA-PEO has provided a binary solution composed of 75 software modules and 425 hardware modules corresponding to fitness equal to 25240. BFA has found a fitness equal to 26723 for a binary vector composed of 7 software modules and 493 hardware modules while NDBE has produced a fitness equal to 26800 for a binary configuration composed of 27 software modules and 473 hardware modules.

As can be seen, BFA-PEO performs significantly better than all other algorithms. The BFA-PEO developed by incorporating a novel probability

Methods / ***Nodes***	***GA***	***BPSO***	***SaBDE***	***DDE***	***BDE***	***NBDE***	***NMBDE***	*NBBFA*	***BFA***	***BFA-PEO***
100	***5147***	*7020*	*7696*	*5532*	*6818*	*5357*	*6830*	*5600*	*5302*	***5147***
200	*10331*	*15265*	*16382*	*10848*	*13839*	*10703*	*14592*	*12209*	*10724*	***10283***
300	*16125*	*23385*	*23989*	*16124*	*21246*	*16065*	*21777*	*18170*	*16060*	***15116***
400	*22598*	*31293*	*32519*	*21430*	*29503*	*21278*	*30656*	*25079*	*21344*	***20214***
500	*29963*	*39454*	*41271*	*26926*	*37232*	*26800*	*38169*	*31330*	*26723*	***25240***
600	*37613*	*48451*	*49400*	*32302*	*46031*	*32285*	*47105*	*37630*	*32216*	***30310***
700	*43854*	*56512*	*57934*	*37530*	*53577*	*37503*	*54915*	*45045*	*37370*	***35180***
800	*52450*	*64525*	*67245*	*42690*	*61918*	*42587*	*61940*	*51272*	*42576*	***40138***
900	*61002*	*73168*	*74058*	*48270*	*70507*	*48163*	*71117*	*58445*	*48247*	***45583***
1000	*69048*	*80771*	*82224*	*53358*	*77814*	*53312*	*78009*	*64545*	*53307*	***50475***

Table 3: Simulation Results.

Iterations \ Algorithms		GA	BPSO	SaBDE	DDE	BDE	NBDE	NMBDE	NBBFA	BFA	BFA-PEO
10	Best	40146	40819	41367	26926	40754	36601	40998	32228	26876	**26509**
	Worst	40741	41367	41367	26926	40754	36601	40998	33456	27217	27135
	Mean	40392,77	41165,17	41367	26926	40754	36601	40998	32837,03	26977,2	26802,43
	St. Dev.	29607,25	418,43	0	0	0	0	0	99317,70	95,13	35682,58
40	Best	37543	40584	41367	26926	39108	28304	40482	32094	26842	**25431**
	Worst	37700	41363	41367	26926	39108	28304	40482	33148	26956	25579
	Mean	37675,4	4103303	41367	26926	39108	28304	40482	32692,63	26891,2	25485,1
	St. Dev.	1542,64	538,33	0	0	0	0	0	66005,77	36,64	1547,49
80	Best	35841	40275	41367	26926	38866	27554	40336	32025	26831	**25301**
	Worst	35934	41355	41367	26926	38866	27554	40336	32991	26924	25388
	Mean	35909,6	40921,2	41367	26926	38866	27554	40336	32534,4	26855,55	25352,1
	St. Dev.	818,92	736,41	0	0	0	0	0	58573,11	28,32	390,557
120	Best	34445	40100	41343	26926	38866	27118	39350	31917	26806	**25285**
	Worst	34687	41347	41343	26926	38866	27118	39350	32759	26881	25337
	Mean	34654,33	40784,5	41343	26926	38866	27118	39350	32366,33	26849,2	25310,43
	St. Dev.	2209,62	788,11	0	0	0	0	0	40742,89	20,25	153,31
160	Best	33181	39775	41343	26926	38605	27036	39350	31680	26788	**25244**
	Worst	33316	41327	41343	26926	38605	27036	39350	32649	26838	25305
	Mean	33288,5	40784,4	41343	26926	38605	27036	39350	32295,4	26818,65	25279,27
	St. Dev.	1929,65	1104,47	0	0	0	0	0	53844,97	12,39	169,86
200	Best	32255	39713	41343	26926	38071	26917	38953	31655	26767	**25244**
	Worst	32398	41297	41343	26926	38071	26917	38953	32649	26827	25283
	Mean	32338,43	40883,1	41343	26926	38071	26917	38953	32252,5	26808,2	25264,57
	St.Dev.	13338,25	1294,31	0	0	0	0	0	62370,72	18,61	121,65
240	Best	30687	39356	41343	26926	38020	26873	38368	31655	26747	**25240**
	Worst	31482	41231	41343	26926	38020	26873	38953	32649	26806	25266
	Mean	31209,47	40643,4	41343	26926	38020	26873	38758	32203,8	26786,3	25252,97
	St. Dev.	136507,3	1413,75	0	0	0	0	76050	52714,03	18,88	45,437
280	Best	30241	39356	41343	26926	38020	26842	38335	31649	26723	**25240**
	Worst	30737	41228	41343	26926	38020	26873	38368	34030	26800	25255
	Mean	30600,9	40437,2	41343	26926	38020	26862,67	38357	33229,1	26763,3	25247,937
	St. Dev.	36528,16	1294,93	0	0	0	213,56	242	311068,42	27,45	13,867
300	Best	29963	39356	41271	26926	37232	26800	38169	31330	26723	**25240**
	Worst	30466	40790	41343	26926	38020	26842	38335	32649	26800	25251
	Mean	30267,27	40083,4	41319	26926	37757,33	26828	38279,67	31983	26737,15	25244,97
	St. Dev.	46573,86	912,69	1152	0	137987,57	392	6123,55	158631,93	17,45	12,56

Table 4: Simulation Results for node=500.

estimation operator (PEO) based on the distribution of estimation algorithm was found to perform better than other algorithms because of an improved balance between exploration and exploitation.

In previous sections, the authors [56, 57] found that the firefly algorithm is superior to PSO and GA in terms of efficiency and success rate. According to Table 3, our results provided by the three variants of FA confirm their claim.

In Table 4, we report the best, worst, Mean, and standard deviation of the evaluation function values over 50 runs. From this table, it is clear that BFA-PEO, BFA and DDE converge quickly to good solutions. At the beginning, from the first ten iterations, both BFA-PEO, BFA, and DDE algorithms have a fast improvement of their objective function. The best cost function has been reduced to almost half. It has improved from 41897 to around 26000 for the three algorithms mentioned above. Then, BFA-PEO fitness continued to decrease until it reached the value of 25240 which represents the best result of all the algorithms. We can see that there is no change in the cost function for DDE and saBDE. Their fitness remains fixed regardless of the number of iterations, while the other methods continue to slightly decrease their function. NBDE does the exception and produces finally, at the 300 iterations, a promising solution whose fitness is equal to 26800. In summary, the BFA-PEO always provides the best results.

5.2 Second experiment

We consider the problem that consists of minimizing the objective function f2(x) with the respect of the Gcmax constraint. f2(x) can be formulated as the follows:

$$Minimize\ f2(x) = Gc$$

$$Subject\ to$$

$$\begin{cases} Gc \leq Gcmax \\ Pc \leq Pcmax \\ Ccom \leq Ccommax \end{cases}$$

Table 5 outlines the results of NBBFA, BFA, BFA-PEO, GA, BPSO, SaBDE, DDE, BDE, NBDE, and NMBDE. The best results are highlighted in bold. As it may be observed, the proposed BFA-PEO algorithm always performs superior to other meta-heuristics. We can see easily the big gap between the values provided by BFA-PEO and the others. For example, for node 1000, the cost function is 11380 for the BFA-PEO, while, for DDE which comes in the second position, the value of the objective function is 58612. Too, we notice

Methods / Nodes	GA	PSO	SaBDE	DDE	BDE	NBDE	NMBDE	NBBFA	BFA	BFA-PEO
100	38287	60556	5782	5782	5782	5782	4041	26172	5782	**1137**
200	261227	271957	81866	11896	17352	11896	17428	137960	11896	**2362**
300	576130	629696	334259	18188	79662	18188	38143	370402	18188	**3561**
400	1142102	1118569	764473	23774	247164	23774	150353	674114	23774	**4602**
500	1727779	1754802	1306480	29624	261859	29624	424189	1095953	29624	**5820**
600	2566649	2549646	2031003	35330	635326	35330	603421	1662421	35330	**6996**
700	3497128	3461190	2827546	41500	1067305	41500	793618	2205311	41500	**7955**
800	4586485	4571337	3898644	47670	2008235	47670	1693334	2718329	47670	**9075**
900	5831982	5812487	4997017	52854	3305082	52854	2124837	3444275	52854	**10247**
1000	7182356	7167670	6376675	58612	4409207	58612	2801863	3944907	58612	**11380**

Table 5: Simulation Results for objective function f2.

that the algorithm is always followed by the other two algorithms BFA and NBDE. The results produced by our BFA-PEO are very logical since, according to the equations used to randomly generate the cost metrics, the algorithm must always provide solutions composed of software blocks only, while the other meta-heuristics give purely hardware or mixed solution. BFA and NBDE provide the same results representing the fitness of purely hardware solutions. The recent NBBFA based on using the Sigmoid function provides poor results and is positioned with the latest algorithms. The superior results mean that the proposed approach can tackle the Hw/Sw partitioning problems efficiently.

6 Conclusion

As the standard FA operates in the continuous space, this paper presents for the first time a binary FA algorithm to tackle the Hw/Sw partitioning problems. The metrics and the constraints are integer numbers generated randomly. The performance of BFA-PEO (Binary Firefly algorithm with Probability Estimation Operator) has then been compared with that of the existing approaches like saBDE, DDE, etc., taking into account the quality of the solution. This type of performance comparison has not been attempted so far on the said problem. The proposed probability estimation operator enables BFA to manipulate binary-valued solutions directly and effectively preserve the diversity of the population and enhance the global search ability. Table 3 and Table 5 summarize the conclusions we have drawn after executing and comparing the different system partitioning methods. A future study could extend the system model to encompass other quality attributes. We will demonstrate the effectiveness of our algorithm on some practical examples.

Acknowledgement

The authors would like to thank the DGRSDT (General Directorate of Scientific Research and Technological Development) - MESRS (Ministry of Higher Education and Scientific Research), ALGERIA, for the financial support of Embedded System Laboratory (LASE).

References

[1] J. Teich(2012). Hardware/Software Codesign: The Past, the Present, and Predicting the Future. *Proceedings of the IEEE*,Vol. 100,pp. 1411–1430. https://doi.org/10.1109/jproc.2011.2182009

[2] F. VAHID (2009). What is hardware/software partitioning? *ACM SIGDA newsletter*, New York, NY. v. 39, n. 6, pp. 4-7. https://doi.org/10.1145/1862900.1862901

[3] W. Jigang, T. Srikanthan, and G. Chen (2010). Algorithmic aspects of hardware/software partitioning: 1D search algorithms. *Institute of Electrical and Electronics Engineers. Transactions on Computers*, vol. 59, no. 4, pp. 532–544. https://doi.ieeecomputersociety.org/10.1109/TC.2009.173

[4] P. Arato, S. Juhasz, Z.A. Mann and A. Orban(2003). Hardware/software partitioning in embedded system design. *In Proc. of the IEEE Int. Symposium on Intelligent Signal Processing*, pp. 197-202. https://doi.org/10.1109/isp.2003.1275838

[5] S.A Tahaee and A.H. Jahangir (2010). A Polynomial Algorithm for Partitioning Problems. *ACM Trans. Embed. Comput. Syst*., vol.9 (4), pp.1–34. https://doi.org/10.1145/1721695.1721700

[6] X.S. Yang (2008). Nature-inspired meta-heuristic algorithms. *Luniver Press*, UK.

[7] A. H. Gandomi (2014). Interior search algorithm (ISA): a novel approach for global optimization. *ISA transactions* 53, no. 4, pp.1168-1183. https://doi.org/10.1016/j.isatra.2014.03.018

[8] X.S. Yang (2012). Efficiency analysis of swarm intelligence and randomization techniques. *Journal of computational and theoretical Nanoscience*, vol.9 (2), pp.189-198. https://doi.org/10.1166/jctn.2012.2012

[9] I. Fister, Jr. I. Fister, X.-S. Yang and J. Brest (2013). A comprehensive review of firefly algorithms. *Swarm and Evolutionary Computation*, 13, 34-46. https://doi.org/10.1016/j.swevo.2013.06.001

[10] H. Wang, W. Wang, and S. Xiao (2019). A survey of firefly algorithm. *Journal of Nanchang Institute of Technology*, vol. 38, no. 4, pp. 71–77.

[11] S. Severin and J. Rossmann (2012). A comparison of different metaheuristic algorithms for optimizing

blended PTP movements for industrial robots. *Intelligent robotics and applications* , pp. 321-330. https://doi.org/10.1007/978-3-642-33503-7_32

[12] S. Delir, A. Foroughi-Asl and S. Talatahari (2019). A Hybrid Charged System Search – Firefly Algorithm For Optimization Of Water Distribution Networks. *International Journal of Optimization in Civil Engineering*, vol.9 (2), pp. 273-290. https://www.sid.ir/en/journal/JournalListPaper.aspx?ID=282331

[13] C. Ren, J. Zhao, L. Chen, and Y. Huang (2018). Application of Firefly Algorithm in Scheduling Optimization of Combined Cooling, Heating and Power with Multiple Objectives. *Chemical Engineering Transactions,* vol.67, pp.829-834. DOI: 10.3303/CET1867139

[14] M. A. Nemmich , F. Debbat (2020). Hybrid Bees Approach Based on Improved Search Sites Selection by Firefly Algorithm for Solving Complex Continuous Functions. *Informatica* 44 (2020) 183–198. https://doi.org/10.31449/inf.v44i2.2385

[15] M.-H. HORNG (2012). Vector quantization using the firefly algorithm for image compression. *Expert Syst. Appl*., vol. 39 (1), pp. 1078–1091. https://doi.org/10.1016/j.eswa.2011.07.108

[16] L. F. F. MIGUEL (2012). Shape and size optimization of truss structures considering dynamic constraints through modern meta-heuristic algorithms. *Expert Systems with Applications*, vol. 39 (10), pp. 9458–9467. https://doi.org/10.1016/j.eswa.2012.02.113

[17] M.K. Sayadi, A. Hafezalkotob and S.G. Jalali Naini (2013). Firefly-inspired algorithm for discrete optimization problems: An application to manufacturing cell formation. *Journal of Manufacturing Systems*, vol.32 (1), pp. 78-84. https://doi.org/10.1016/j.jmsy.2012.06.004

[18] M.K. Sayadi, R. Ramezanian, and N. Ghaffari-Nasab (2010). A discrete firefly metaheuristic with local search for makespan minimization in permutation flow shop scheduling problems. *International Journal of Industrial Engineering Computations*, Vol.1, No. 1, pp.1-10. https://doi.org/10.5267/j.ijiec.2010.01.001

[19] K. Durkota (2011). Implementation of a Discrete Firefly Algorithm for the QAP Problem within the Seage Framework. *BSc Thesis, Faculty of Electrical Engineering, Czech Technical University.*

[20] S. Palit, S. Sinha, , M. Molla, , A. Khanra and M. Kule (2011). A cryptanalytic attack on the knapsack cryptosystem using binary firefly algorithm. *In : the second international conference on computer and communication technology, IEEE,* pp.428-432. https://doi.org/10.1109/iccct.2011.6075143

[21] R. Falcon, M. Almeida and A. Nayak (2011). Fault identification with binary adaptive fireflies in parallel and distributed systems. *In: IEEE congress on evolutionary computation (CEC-2011), IEEE,* pp.1359-1366. https://doi.org/10.1109/cec.2011.5949774

[22] K. Chandrasekaran and S. Simon (2012). Network and reliability constrained unit commitment problem using binary real coded firefly algorithm. *International journal of electrical power & energy systems,* vol. 43(1), pp. 921-932. https://doi.org/10.1016/j.ijepes.2012.06.004

[23] Khadwilard, S. Chansombat, T. Thepphakorn, W. Chainate, and P. Pongcharoen (2012). Application of firefly algorithm and its parameter setting for job shop scheduling. *The Journal of Industrial Technology*, Vol. 8, No. 1, pp.49-58. http://www.ojs.kmutnb.ac.th/index.php/joindtech/article/view/4399

[24] S. KARTHIKEYAN, P. Asokan, S. Nickolas and T. Page (2015). A hybrid discrete firefly algorithm for solving multi-objective flexible job shop scheduling problems. *International Journal of Bio-Inspired Computation,* vol. 7 (6), pp. 386-407. http://dx.doi.org/10.1504/IJBIC.2015.073165

[25] B. Crawford, R. Soto, M. Olivares-Suarez, and W. Palma (2014). A Binary Coded Firefly Algorithm That Solves the Set Covering Problem. *In Advances in Intelligent Systems and Computing,*pp. 65- 73 https://doi.org/10.1007/978-3-319-06740-7_6

[26] R. F. Najeeb and B. N. Dhannoon (2018). A Feature Selection Approach Using Binary Firefly Algorithm For Network Intrusion Detection System. *ARPN Journal of Engineering and Applied Sciences.* VOL. 13, NO. 6. ISSN 1819-6608 http://www.arpnjournals.org/jeas/research_papers/rp_2018/jeas_0318_6935.pdf

[27] R.R. Rajalaxmi, E. Gothai, .R. Thamilselvan, P. Gokila Bindha, and P.Natesan (2019). Naïve Bayes guided Binary Firefly Algorithm for Gene Selection in Cancer Classification. *International Journal of Recent Technology and Engineering (IJRTE)* ISSN: 2277-3878, Volume-8 Issue-4.pp. 7405 -7409 https://doi.org/10.35940/ijrte.d5308.118419

[28] M. Lin, F. Liu, H. Zhao and J. Chen (2020). A Novel Binary Firefly Algorithm for the Minimum Labeling Spanning Tree Problem. Computer Modeling in Engineering & Sciences Tech Science Press. DOI :10.32604/cmes.2020.09502

[29] S. L. Tilahun and J. M. T. Ngnotchouye (2016). Firefly Algorithm for optimization problems with non-continuous variables: *A Review and Analysis. Preprint submitted to Elsevier.* https://arxiv.org/abs/1602.07884v1

[30] V. Kumar and D. Kumar (2021). A Systematic Review on Firefly Algorithm: Past, Present, and Future. *Archives of Computational Methods in Engineering,* Vol. 28, pp.3269–3291 https://doi.org/10.1007/s11831-020-09498-y

[31] S. A. Edwards, L. Lavagno, E. A. Lee et al.(1997). Design of Embedded Systems: Formal Models Validation, and Synthesis. *Proceedings of the IEEE,* vol.85 (3), pp.366-390. https://doi.org/10.1109/5.558710

[32] B. Mei, P. Schaumont and S.Vernalde (2000). A hardware-software partitioning and scheduling

algorithm for dynamically reconfigurable embedded systems. *In Proceedings of ProRISC*. Citeseer, pp. 405–411.
http://rijndael.ece.vt.edu/schaum/papers/2000prorisc.pdf

[33] R. Ernst, J. Henkel and T. Benner (2002). Hardware-software cosynthesis for microcontrollers. *Readings in hardware/software co-design*, pp. 18–29.
https://doi.org/10.1016/b978-155860702-6/50004-1

[34] J. Wu, T. Srikanthan and C. Yan (2008). Algorithmic aspects for power-efficient hardware/software partitioning. *Math Comput Simul.* , vol.79 (4), pp.1204–1215.
https://doi.org/10.1016/j.matcom.2007.09.003

[35] R.S. Rajakumari, K. Hariharan, R. Manikandan and K.R. Sekar (2018). A Survey on various method used in Hardware and Software Partitioning. *International Journal of Pure and Applied Mathematics*, Vol. 118 No. 18, pp.2365-2385. ISSN: 1311-8080 (printed version); ISSN: 1314-3395 (on-line version)
url: http://www.ijpam.eu

[36] Y. Jing, J. Kuang, J. Du, and B. Hu (2014). Application of improved simulated annealing optimization algorithms in hardware/software partitioning of the reconfigurable system-on-chip. *Communications in Computer and Information Science, vol*. 405, pp. 532– 540.
https://doi.org/10.1007/978-3-642-53962-6_48

[37] J. Wu, P. Wang, S.-K. Lam, and T. Srikanthan(2013). Efficient heuristic and tabu search for hardware/software partitioning. *The Journal of Supercomputing*, vol. 66, no. 1, pp. 118–134.
https://doi.org/10.1007/s11227-013-0888-9

[38] M.M. Shehab, A.T. Khader and M.A. Al-Betar (2016). New selection schemes for particle swarm optimization. *In Proceedings ,the 7th International Conference on Information Technology.*
http://dx.doi.org/10.15849/icit.2015.0003

[39] G. Li, J. Feng, J. Hu, C. Wang and D. Qi (2014). Hardware/Software Partitioning Algorithm Based on Genetic Algorithm. *Journal of Computers*, Vol. 9, No. 6.
https://doi.org/10.4304/jcp.9.6.1309-1315

[40] M. Riabi, Y. Manai, and J.Haggège (2015). Hardware/Software partitioning approach for embedded system design based on Genetic Algorithm. *International Journal of Scientific Research & Engineering Technology (IJSET)* ISSN: 2356-5608, Vol.3, issue 3, pp.20-25.
ipco-co.com/IJSET/ACECS2015/5.pdf

[41] S.-A. Li, C.-C. Hsu, C.-C. Wong, and C.-J. Yu(2011). Hardware/software co-design for particle swarm optimization algorithm. *Information Sciences*, vol. 181, no. 20, pp. 4582 4596.
https://doi.org/10.1016/j.ins.2010.07.017

[42] M.B. Abdelhalim , S.E.-D. Habib (2011). An integrated high-level hardware/software partitioning methodology. *Des. Autom. Embeded System*, vol.15, pp.19–50. DOI 10.1007/s10617-010-9068-9

[43] E. M. F. El Houby, N. I. R. Yassin and S. Omran (2017). A Hybrid Approach from Ant Colony Optimization and K-nearest Neighbor for Classifying Datasets Using Selected Features. *Informatica* ,vol. 41(4),pp. 495–506.
https://www.informatica.si/index.php/informatica/article/view/1444/1096

[44] Y.-D. Zhang, L.-N. Wu, G. Wei, H.-Q. Wu, and Y.-L. Guo(2009). Hardware/software partition using adaptive ant colony algorithm. *Control and Decision*, Issue 9, pp. 1385–1389.

[45] D. Das, M.L. Verma and A. Das (2014). A Differential Evolutionary Approach to Solve the Hardware Software Partitioning Problem. *International Journal of Engineering Research & Technology*, vol.3, issue 7. ISSN: 2278-0181 IJERTV3IS071071

[46] S. Prakasam, M. Venkatachalam, M. Saroja, N. Pradheep, and P. Gowthaman (2016). A novel bat inspired algorithm for hardware-software codesign partitioning. *International Journal of Multidisciplinary Research and Development*. Vol. 3; Issue 3; pp. 88-92; (Special Issue). Online ISSN: 2349-4182
http://www.allsubjectjournal.com

[47] A. Iguider, K. Bousselam, O. Elissati, M. Chami, and A. En-Nouaary (2020). Heuristic algorithms for multi-criteria hardware/software partitioning in embedded systems codesign. *Computers and Electrical Engineering*, vol. 84, pp. 106-210.
https://doi.org/10.1016/j.compeleceng.2020.106610

[48] XH. Yan, FZ. He and YL. Chen (2017). A novel hardware/software partitioning method based on position disturbed particle swarm optimization with invasive weed optimization. *Journal Of Computer Science And Technology*, vol. 32(2), pp. 340–355.
DOI 10.1007/s11390-017-1714-2

[49] W. Shi, J. Wu, S.-k. Lam, and T. Srikanthan (2016). Algorithms for Bi-objective Multiple-choice Hardware/Software Partitioning. *Computers and Electrical Engineering*, vol. 50, pp. 127-142.
https://doi.org/10.1016/j.compeleceng.2016.01.006

[50] L. Li, J. Sun, W. Li, Z. Lv and F. Guan. Hardware/software partitioning based on hybrid genetic and tabu search in the dynamically reconfigurable system (2015). *International J. of Control & Automation*. Vol. 8(1) pp.29–36.
http://dx.doi.org/10.14257/ijca.2015.8.1.03

[51] S. Dimassi, M. Jemai, B. Ouni and A. Mtibaa (2015). Optimization Algorithms for Hardware/Software Partitioning. *International journal of computer science,communication & information Technology (CSCIT)*,vol.2(1),pp.23-27.
ipco-co.com/CSCIT_Journal/papers-CSCIT/CSCIT/CSCIT%20-%20Vol.2%20-%20issue1%20-%202015/5.pdf

[52] L. An, F. Jinfu, L. Xiaolong, and Y. Xiaotian(2010). Algorithm of hardware/software partitioning based on genetic particle swarm optimization. Journal of

Computer-Aided Design & Computer Graphics, vol. 22, no. 6, pp. 927–933. https://doi.org/10.3724/sp.j.1089.2010.10834

[53] G. Li, J. Feng, C. Wang and J. Wang (2014). Improved hardware/software partitioning algorithm based on combination of PSO and TS. *Journal of Comput. Inf. System.* Vol 10(14) ,pp. 5975–5985.

[54] L.Weijia , L. Lanying , S. Jianda , L. Zhiqiang, and F. Guan (2014). Hardware/software partitioning of combination of clustering algorithm and genetic algorithm. *International journal of control and automation*, vol. 7, no. 1,pp 347-356. https://doi.org/10.14257/ijca.2014.7.1.31

[55] X. Zhao, H. Zhang, Y. Jiang, S. Song, X. Jiao, and M. Gu (2013). An Effective Heuristic-Based Approach for Partitioning. *Hindawi Publishing Corporation Journal of Applied Mathematics.* Volume 2013, Article ID 138 037, 8 pages. http://dx.doi.org/10.1155/2013/138037

[56] X.S. Yang (2009). Firefly algorithms for multimodal optimization, Stochastic Algorithms. *Foundations and Applications (SAGA) Lecture Notes in Computer Sciences*, 5792, pp. 169–78. https://doi.org/10.1007/978-3-642-04944-6_14

[57] X.S. Yang (2010). Firefly Algorithm, Levy Flights and Global Optimization. *Research and Development in Intelligent Systems XXVI (Eds M. Bramer, R. Ellis, Petridis), Springer London*, pp. 209-218. https://doi.org/10.1007/978-1-84882-983-1_15

[58] P.Arató, Z. Á. Mann and A. Orbán (2005). Algorithmic aspects of hardware/software partitioning. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 10, no. 1, pp. 136–156. https://doi.org/10.1145/1044111.1044119

[59] K. Yahyaoui (2017). Partitioning and Scheduling Resolution Problems By Bees Mating Strategy. In Dres' Systems. *International Journal of Computing*, vol. 16(2), pp.97-105. https://doi.org/10.47839/ijc.16.2.886

[60] G. Li, J. Feng, C. Wang and J. Wang (2014). Hardware/software partitioning algorithm based on the combination of genetic algorithm and tabu search. *Engineering Review*, vol. 34, no. 2, pp. 151 –160. https://hrcak.srce.hr/122370

[61] T.-Y. Lee, Y.-H. Fan, Y.-M. Cheng, C.-C. Tsai and R.-S. Hsiao (2007). Enhancement of hardware-software partition for embedded multiprocessor FPGA systems. *In Third International Conference on Intelligent Information Hiding and Multimedia Signal Processing, IIHMSP 2007*, vol. 1. IEEE, 2007, pp. 19–22. https://doi.org/10.1109/iihmsp.2007.4457483

[62] P. K. Nath and D.Datta (2014). Multi-objective hardware–software partitioning of embedded systems: A case study of JPEG encoder. *Applied Soft Computing*, vol. 15, pp. 30–41. https://doi.org/10.1016/j.asoc.2013.10.032

[63] E. Sha, L. Wang, Q. Zhuge, J. Zhang and J. Liu(2015). Power efficiency for hardware/software partitioning with time and area constraints on MPSOC. *International Journal of Parallel Programming*, vol.43(3),pp. 381 –402. https://doi.org/10.1007/s10766-013-0283-4

[64] M. Lopez-Vallejo and J. Lopez (2003). On the hardware-software partitioning problem: System modeling and partitioning techniques. *ACM Transactions on Design Automation of Electronic Systems*, vol. 8(3), pp.269–297. http://dx.doi.org/10.1145/785411.785412

[65] N. Govil and S. R. Chowdhury (2015). A High Speed Metaheuristic Algorithmic Approach to Hardware Software Partitioning for Low-cost SoCs. *International Symposium on Rapid System Prototyping (RSP)*, 2015. https://doi.org/10.1109/rsp.2015.7416554

[66] X. Zhao, H. Zhang, Y. Jiang, S. Song, X. Jiao and M.Gu (2013). An Effective Heuristic-Based Approach for Partitioning. *Journal of Applied Mathematics*, Vol. 2013, Article ID 138037, 8 pages. https://doi.org/10.1155/2013/138037

[67] Task Graphs for Free (TGFF v3.0) [EB/OL]. [2012-11-20]. http://z iyang.eecs.umich.edu/ dickrp/tgff/.

[68] A. K.Qin and P. N. Suganthan (2005). Self-adaptive differential evolution algorithm for numerical optimization. *In Proceedings of IEEE Congress on Evolutionary Computation*, vol. 2, pp. 1785–1791. https://doi.org/10.1109/cec.2005.1554904

[69] J. Krause and H. S. Lopes (2013). A Comparison of Differential Evolution Algorithm with Binary and Continuous Encoding for the MKP. *BRICS Congress on Computational Intelligence & 11th Brazilian Congress on Computational Intelligence, 2013.* https://doi.org/10.1109/brics-cci-cbic.2013.70

[70] H. Xingshi and L.Han (2007). A novel binary differential evolution algorithm based on artificial immune system. *IEEE congress on evolutionary computation (CEC 2007)*, pp.2267-2272. https://doi.org/10.1109/cec.2007.4424753

[71] A. W. Mohamed (2016). A New Modified Binary Differential Evolution Algorithm and its Applications. *Applied Mathematics & Information Sciences*, vol.10, No. 5, pp. 1965-1969. https://doi.org/10.18576/amis/100538